PCE Working Group   H. Chen
Internet-Draft   Huawei Technologies
Intended status: Standards Track   M. Toy
Expires: May 4, 2017

                                                         L. Liu
                                                        Fujitsu
                                                         V. Liu
                                                   China Mobile
                                               October 31, 2016

PCE Hierarchical SDNs
draft-chen-pce-h-sdns-01

Abstract

   This document presents extensions to the Path Computation Element
   Communication Protocol (PCEP) for supporting a hierarchical SDN
   control system, which comprises multiple SDN controllers controlling
   a network with a number of domains.

Status of This Memo

Copyright Notice

   to this document. Code Components extracted from this document must
   include Simplified BSD License text as described in Section 4.e of
   the Trust Legal Provisions and are provided without warranty as
   described in the Simplified BSD License.

Table of Contents

[1]. Introduction

   A domain is a collection of network elements within a common sphere
   of address management or routing procedure which are operated by a
   single organization or administrative authority. Examples of such
   domains include IGP (OSPF or IS-IS) areas and Autonomous Systems.

   For scalability, security, interoperability and manageability, a big
   network is organized as a number of domains. For example, a big
   network running OSPF as routing protocol is organized as a number of
   OSPF areas. A network running BGP is organized as multiple
   Autonomous Systems, each of which has a number of IGP areas.

   The concepts of Software Defined Networks (SDN) have been shown to
   reduce the overall network CapEx and OpEx, whilst facilitating the
   deployment of services and enabling new features. The core
   principles of SDN include: centralized control to allow optimized
   usage of network resources and provisioning of network elements
   across domains.

   For a network with a number of domains, it is natural to have
   multiple SDN controllers, each of which controls a domain in the
   network. To achieve a centralized control on the network, a
   hierarchical architecture of controllers is a good fit. At top level
   of the hierarchy, it is a parent controller that is not a child
   controller. The parent controller controls a number of child
   controllers. Some of these child controllers are not parent
   controllers. Each of them controls a domain. Some other child
   controllers are also parent controllers, each of which controls
   multiple child controllers, and so on.

   This document presents extensions to the Path Computation Element
   Communication Protocol (PCEP) for supporting a hierarchical SDN
   control system, which comprises multiple SDN controllers controlling
   a network with a number of domains.

[2]. Terminology

   The following terminology is used in this document.

   ABR: Area Border Router. Router used to connect two IGP areas
      (Areas in OSPF or levels in IS-IS).

   ASBR: Autonomous System Border Router. Router used to connect
      together ASes of the same or different service providers via one
      or more inter-AS links.

BN: Boundary Node. A boundary node is either an ABR in the context
   of inter-area Traffic Engineering or an ASBR in the context of
   inter-AS Traffic Engineering. A Boundary Node is also called an
   Edge Node.

Entry BN of domain(n): a BN connecting domain(n-1) to domain(n)
   along the path found from the source node to the BN, where
   domain(n-1) is the previous hop (or upstream) domain of domain(n).
   An Entry BN is also called an in-BN or in-edge node.

Exit BN of domain(n): a BN connecting domain(n) to domain(n+1) along
   the path found from the source node to the BN, where domain(n+1)
   is the next hop (or downstream) domain of domain(n). An Exit BN
   is also called a out-BN or out-edge node.

Source Domain: For a tunnel from a source to a destination, the
   domain containing the source is the source domain for the tunnel.

Destination Domain: For a tunnel from a source to a destination, the
   domain containing the destination is the destination domain for
   the tunnel.

Source Controller: A controller controlling the source domain.

Destination Controller: A controller controlling the destination
   domain.

Parent Controller: A parent controller is a controller that
   communicates with a number of child controllers and controls a
   network with multiple domains through the child controllers. A
   PCE can be enhanced to be a parent controller.

Child Controller: A child controller is a controller that
   communicates with one parent controller and controls a domain in a
   network. A PCE can be enhanced to be a child controller.

Exception list: An exception list for a domain contains the nodes in
   the domain and its adjacent domains that are on the shortest path
   tree (SPT) that the parent controller is building.

GTID: Global Tunnel Identifier. It is used to identify a tunnel in
   a network.

PID: Path Identifier. It is used to identify a path for a tunnel in
   a network.

   Inter-area TE LSP: a TE LSP that crosses an IGP area boundary.

   Inter-AS TE LSP: a TE LSP that crosses an AS boundary.

   LSP: Label Switched Path

   LSR: Label Switching Router

   PCC: Path Computation Client. Any client application requesting a
      path computation to be performed by a Path Computation Element.

   PCE: Path Computation Element. An entity (component, application,
      or network node) that is capable of computing a network path or
      route based on a network graph and applying computational
      constraints.

   PCE(i): a PCE with the scope of domain(i).

   TED: Traffic Engineering Database.

   This document uses terminology defined in [RFC5440].

3. Conventions Used in This Document

   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].

4. Requirements

   This section summarizes the requirements for Hierarchical SDN Control
   System (need more text here).

5. Overview of Hierarchical SDN Control System

   The Figure below illustrates a hierarchical SDN control system.
   There is one Parent Controller and four Child Controllers: Child
   Controller 1, Child Controller 2, Child Controller 3 and Child
   Controller 4.

```
                        +------------------+
                        | Parent Controller |
                        +--+---------+----+-+
                       _/|           \   \____
                     _/  |            \    \____
                   _/    |             \      \__
                 __/     |   +---------+---------+   \
               __/       |   |Child Controller 3 |    |
              /          |   +------------------+     |
    +---------+---------+ |      /        \           |
    |Child Controller 1 | |    .---. .---,\           |
    +------------------+  |   (     '     ')          |
         /        \       |   (  Domain 3  )          |
      .---. .---,\        |   (           )  +---------+---------+
     (     '     ')       |    '-o-.--o)   |Child Controller 4 |
     (  Domain 1  )       |          |     +------------------+
     (           )        |          |        /          \____
       '-o-.---)  +--------+---------+  \     /            \   \____
          |       |Child Controller 2 |   \   /\  .---. .---+      \
          |       +------------------+    \  | \(      '   |'.---. |
          |         /        \____         \_ |---\ Domain 4 |      '+,
          \        /          \   \____       (o    \         |       | )
           \      /\  .---. .---+       \      (     |         |       o)
            \     | \(      '   |'.---. |      (     |         |        )
             \    |---\ Domain 2 |      '+.    (   o         o    .-'
          \____(o    \          |        | )     '              )
               (     |          |     o)-------o---._.-.-----)
               (     |          |         )
               (   o         o    .-'
                  '               )
                   '---._.-.-----)
```
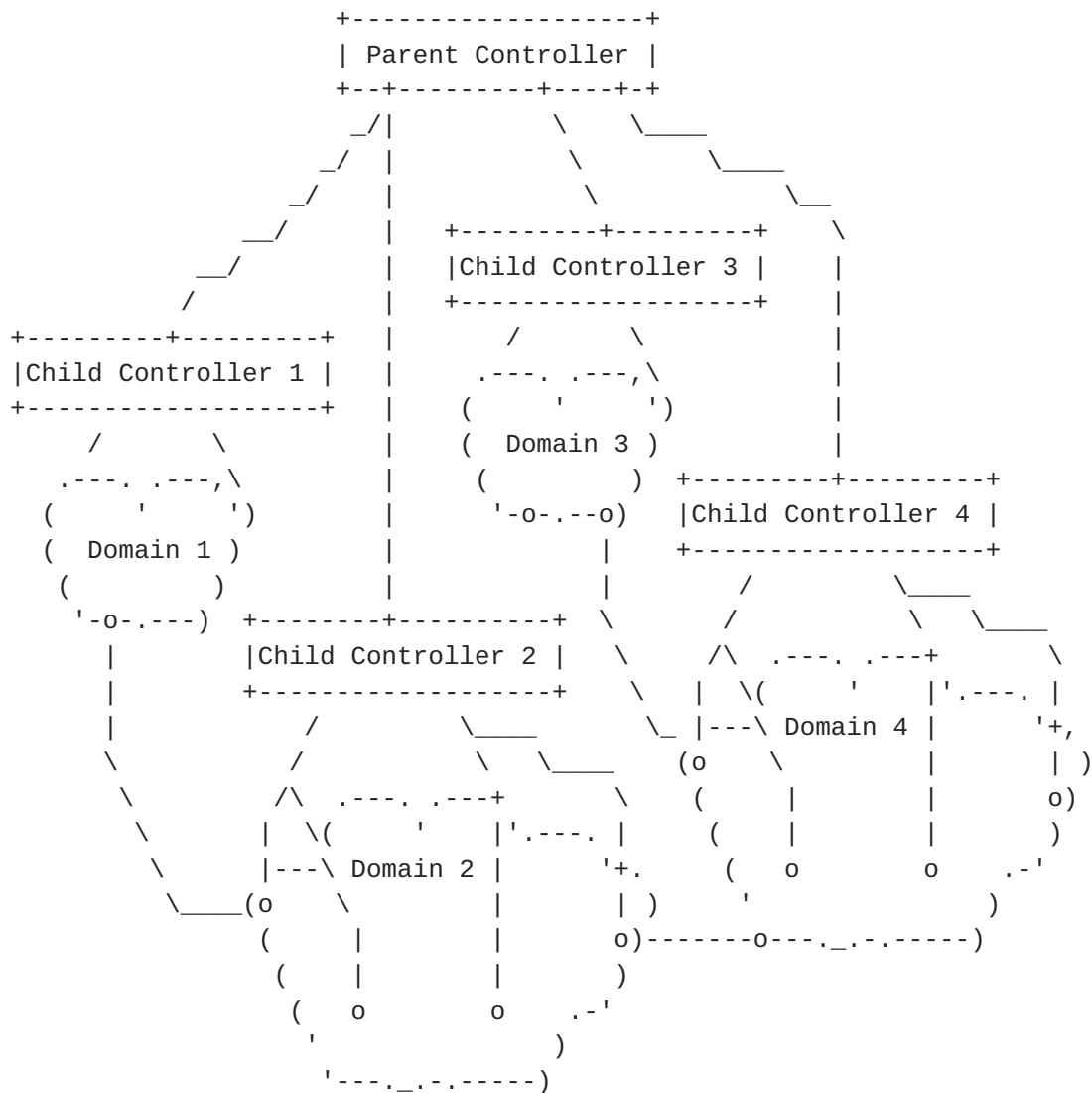
   The parent controller communicates with these four child controllers
   and controls them, each of which controls (or is responsible for) a
   domain. Child controller 1 controls domain 1, Child controller 2
   controls domain 2, Child controller 3 controls domain 3, and Child
   controller 4 controls domain 4.

   One level of hierarchy of controllers is illustrated in the figure
   above. There is one parent controller at top level, which is not a
   child controller. Under the parent controller, there are four child
   controllers, which are not parent controllers.

   In a general case, at top level there is one parent controller that
   is not a child controller, there are some controllers that are both
   parent controllers and child controllers, and there are a number of
   child controllers that are not parent controllers. This is a system

of multiple levels of hierarchies, in which one parent controller
controls or communicates with a first number of child controllers,
some of which are also parent controllers, each of which controls or
communicates with a second number of child controllers, and so on.

Considering one parent controller and its child controllers, each of
the child controllers controls a domain and has the topology
information on the domain, the parent controller does not have the
topology information on any domain controlled by a child controller
normally. This is called parent without domain topology.

In some special cases, the parent controller has the topology
information on a region consisting of the domains controlled by its
child controllers. In other words, the parent controller has the
topology information on the domains controlled by its child
controllers and the topology/inter-connections among these domains.
This is called parent with domain topology.

The parent controller receives requests for creating end to end
tunnels from users or applications. For each request, the parent
controller is responsible for obtaining a path for the tunnel and
creating the tunnel along the path.

For parent without domain topology, the parent controller asks each
of its related child controllers to compute path segments from an
entry boundary node to exit boundary nodes in the domain it controls
or path segments from an exit boundary node in its domain to entry
boundary nodes of other adjacent domains just using the inter-domain
links attached to the exit boundary node. The details of the
segments are hidden from the parent, which sees each of the segments
as a link from a boundary node to another boundary node with a cost.
The parent controller builds a shortest path tree (SPT) using the
path segments computed as links to get the end to end path and then
creates the tunnel along the path by asking its related child
controllers.

The end to end path does not have any details from the parent's point
of view. It can be considered as a sequence of domains containing
the shortest path. Along this sequence of domains, the details of
the end to end path can be obtained. And then the tunnel along the
path with details can be created.

For parent with domain topology, the parent controller computes a
path for the tunnel using the topology information on the domains
controlled by its child controllers. And then it creates the tunnel
along the path computed through asking its related child controllers.

6. **Extensions to PCEP**

   This section describes the extensions to PCEP for a Hierarchical SDN
   Control System (HSCS). The extensions include the definition of a
   new flag in the RP object, a global tunnel identifier (GTID), a path
   identifier (PID), a list of path segments and an exception list in
   the PCReq and PCRep message.

6.1. **Capability Discovery**

   During a PCEP session establishment between two PCEP speakers (PCE or
   PCC), each of them advertises its capabilities for HSCS through the
   Open Message with the Open Object containing a new TLV to indicate
   its capabilities for HSCS. This new TLV is called HSCS capability
   TLV. It has the following format.

```
     0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |            Type = TBD1         |             Length            |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                       Capability Flags                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                      (Optional Sub-TLVs)                      |
     ~                                                               ~
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The type of the TLV is TBD1. It has a length of 4 octets plus the
   size of optional Sub-TLVs. The value of the TLV comprises a
   capability flags field of 32 bits, which are numbered from the most
   significant as bit zero. Each bit represents a capability.

   o PC (Parent Controller - 1 bit): Bit 0 is used as PC flag. It is
      set to 1 indicating a parent controller.

   o CC (Child Controller - 1 bit): Bit 1 is used as PC flag. It is
      set to 1 indicating a child controller.

   o PS (Path Segments - 1 bit): Bit 2 is used as PS flag. It is set
      to 1 indicating support for computing path segments for HSCS

   o TS (Tunnel Segment - 1 bit): Bit 3 is used as TS flag. It is set
      to 1 indicating support for creating tunnel segment for HSCS

   o ET (End to end Tunnel - 1 bit): Bit 4 is used as ET flag. It is
      set to 1 indicating support for creation and maintenance of end to
      end LSP tunnels

6.2. New Messages for Hierarchical SDN Control System

   This section describes the contents and semantics of the new
   messages, and presents a few of different encodings for the messages.

   There are a number of new messages for supporting HSCS. These new
   messages can be encoded in a few of ways as follows:

   o To use a new type at top level for each of the new messages. This
      is called individual encoding.

   o To use a new type at top level for each group of the new messages
      and a option/operation/sub-type value for every message in the
      group. This is called group encoding.

   o To use/re-use existing messages and a value of options/operations
      for each new message in an existing message. This is called
      embedded encoding.

   o To combine the ways above. This is called mixed encoding.

   Various types of messages for supporting HSCS are listed below. Note
   that many new messages may not be needed for some procedures/options.
   For example, four messages Request and Reply for Removing Path
   Segments and Request and Reply for Keeping Path Segments are not
   needed if path segments computed are not stored/remembered by a child
   controller. But in this case, the path segment in each domain along
   the end to end path computed needs to be re-computed when a tunnel
   along the path is set up.

   Message for Controller Relation Discovery: It is a message exchanged
      between a parent controller and a child controller for discovering
      their parent-child relation.

   Message for Connections and Accesses Advertisement: It is a message
      that a child controller sends its parent controller to describe
      the connections from the domain it controls to its adjacent
      domains and the access points in the domain to be accessible
      outside of the domain.

   Request for Computing Path Segments: It is a message that a parent
      controller sends a child controller to request the child
      controller for computing path segments in the domain the child
      controller controls.

   Reply for Computing Path Segments: It is a message that a child
      controller sends a parent controller to reply the parent
      controller for a request message for computing path segments after
      receiving the request message from the parent controller for
      computing path segments and computing path segments as requested,
      which normally contains the path segments computed.

   Request for Removing Path Segments: It is a message that a parent
      controller sends a child controller to request the child
      controller for removing the path segments computed by the child
      controller and stored in the child controller.

   Reply for Removing Path Segments: It is a message that a child
      controller sends a parent controller to reply the parent
      controller for a request message for removing a set of path
      segments after receiving the request message from the parent
      controller for removing path segments and removing the path
      segments as requested, which normally contains a status of
      removing path segments.

   Request for Keeping Path Segments: It is a message that a parent
      controller sends a child controller to request the child
      controller for keeping a set of path segments computed by the
      child controller and stored in the child controller.

   Reply for Keeping Path Segments: It is a message that a child
      controller sends a parent controller to reply the parent
      controller for a request message for keeping path segments after
      receiving the request message from the parent controller for
      keeping path segments and keeping the path segments as requested,
      which normally contains a status of keeping path segments.

   Request for Creating Tunnel Segment: It is a message that a parent
      controller sends a child controller to request the child
      controller for creating tunnel segments related to the domain the
      child controller controls.

   Reply for Creating Tunnel Segment: It is a message that a child
      controller sends a parent controller to reply the parent
      controller for a request message for creating tunnel segment after
      receiving the request message from the parent controller for
      creating tunnel segment and creating tunnel segment as requested,
      which normally contains a status of creating tunnel segment and a
      label and an interface.

   Request for Removing Tunnel Segment: It is a message that a parent
      controller sends a child controller to request the child
      controller for removing the tunnel segment created by the child
      controller.

   Reply for Removing Tunnel Segment: It is a message that a child
      controller sends a parent controller to reply the parent
      controller for a request message for removing tunnel segment after
      receiving the request message from the parent controller for
      removing tunnel segment and removing the tunnel segment as
      requested, which normally contains a status of removing tunnel
      segment.

6.2.1. Contents of Messages

   This section describes the contents in each of the messages and gives
   the format of each of messages in individual encoding, which is the
   same as in group encoding. Some of the objects in the messages are
   defined in the following sections.

6.2.1.1. Message for Controller Relation Discovery

   A message for controller relation discovery is exchanged between a
   parent controller and a child controller for discovering their
   parent-child relation.

   A message for controller relation discovery (CRDis message for short)
   sent from a local controller to a remote controller comprises:

   o Local controller attributes

   o Remote controller attributes after the local controller receives
      the remote controller attributes from a remote end and determines
      that the relation between the local controller and the remote
      controller can be formed.

   The format of the CRDis message is as follows:

     <CRDis Message> ::= <Common Header>
                         <CRP>
                         <Local-Controller>
                         [<Remote-Controller>]

   where CRP (Controller Request Parameters) object is defined in
   section Objects and TLVs.

6.2.1.2. **Message for Connections and Accesses Advertisement**

   After a child controller discovers its parent controller, it sends
   its parent controller a message for connections and accesses
   advertisement.

   A message for connections and accesses advertisement (CAAdv message
   for short) from a child controller comprises:

   o Inter-domain links from the domain the child controller controls
      to its adjacent domains.

   o The addresses in the domain to be accessible to the outside of the
      domain.

   o Attributes of each of the boundary nodes of the domain.

   The format of the CAAdv message is as follows:

      <CAAdv Message> ::= <Common Header>
                          <CRP>
                          <Inter-Domain-Link-List>
                          [<Access-Address-List>]
      where:
      <Inter-Domain-Link-List> ::= <Inter-Domain-Link>
                                   [<Inter-Domain-Link-List>]
      <Access-Address-List> ::= <Access-Address>
                                [<Access-Address-List>]

6.2.1.3. **Request for Computing Path Segments**

   After receiving a request for creating an end to end tunnel from
   source A to destination Z for a given set of constraints, a parent
   controller allocates a global tunnel identifier (GTID) for the end to
   end tunnel crossing domains and a path identifier (PID) for an end to
   end path to be computed for the tunnel. The parent controller sends
   a request message to each of its related child controllers for
   computing a set of path segments in the domain the child controller
   controls in a special order. The parent controller builds a shortest
   path tree (SPT) using these path segments and obtains a shortest path
   from source A to destination Z that satisfies the constraints.

   Note: The details of the path segments are hidden from the parent,
   which sees each of the segments as a link from one (boundary) node to
   another (boundary) node with a cost. The end to end path does not
   have any details from the parent's point of view, which may be
   considered as a domain path.

A request message for computing path segments (PSReq message for
short) from a parent controller to a child controller comprises:

o The address or identifier of the start-node (saying X) in the
   domain controlled by the child controller. From this node, a
   number of path segments are to be computed.

o The global tunnel identifier (GTID) and the path identifier (PID).
   For the path of the tunnel, a number of path segments are to be
   computed.

o An exception list containing the nodes that are on the SPT and in
   the domain controlled by the child controller or its adjacent
   domains.

o The constraints for the path such as bandwidth constraints and
   color constraints.

o A destination node Z. If Z is in the domain controlled by the
   child controller, the child controller computes a shortest path
   segment satisfying the constraints from node X to node Z within
   the domain.

o Options for computing path segments:

   E: E set to 1 indicating computing a shortest path segment
      satisfying the constraints from node X to each of the edge
      nodes of the domain controlled by the child controller except
      for the nodes in the exception list. E is set to 1 if there is
      not any previous hop of node X in the domain.

After receiving the request message, the child controller computes a
shortest path segment satisfying the constraints from node X to each
of the edge nodes of the domain controlled by the child controller
except for the nodes in the exception list if E is 1. In addition,
it computes a shortest path segment satisfying the constraints from
node X to each of the edge nodes of the adjacent domains except for
the edge nodes in the exception list just using the inter-domain
links attached to node X if node X is an edge node of the domain and
an end point of an inter-domain link.

The format of the PSReq message is as follows:

```
   <PSReq Message> ::= <Common Header>
                       [<svec-list>]
                       <path-segment-request-list>
   where:
     <svec-list>::=<SVEC>[<svec-list>]
     <path-segment-request-list> ::=
                       <path-segment-request>
                       [<path-segment-request-list>]

     <path-segment-request> ::=
                       <CRP>
                       <Start-Node> <Tunnel-ID> <Path-ID>
                       [<Destination>]
                       [<OF>] [<LSPA>] [<BANDWIDTH>]
                       [<metric-list>] [<RRO>[<BANDWIDTH>]] [<IRO>]
                       [<LOAD-BALANCING>]
                       <exception-list>
```

6.2.1.4. Reply for Computing Path Segments

   After receiving a request message from a parent controller for
   computing path segments, a child controller computes the path
   segments as requested in the message and sends the parent controller
   a reply message to reply the request message, which contains the path
   segments computed. The details of the path segments are hidden from
   the parent, which sees each of the path segments as a link with a
   cost.

   A reply message for computing path segments (PSRep message for short)
   comprises:

   o The global tunnel identifier (GTID) and the path identifier (PID).
      For the path of the tunnel, the path segments are computed.

   o The address or identifier of the start-node (saying X) in the
      domain controlled by the child controller. From this node, the
      path segments are computed.

   o For each shortest path segment from node X to node Y computed, the
      address or identifier of node Y and the cost of the shortest path
      segment from node X to node Y.

   The child controller stores the details about every shortest path
   segment computed under the global tunnel identifier (GTID) and the
   path identifier (PID) when it sends the reply message containing the
   path segments to the parent controller.

The child controller may delete the path segments computed for the
global tunnel identifier (GTID) and the path identifier (PID) if it
does not receive any request for keeping them from the parent
controller for a given period of time.

The format of the PSRep message is as follows:

```
   <PSRep Message> ::= <Common Header>
                       <path-segment-reply-list>
   where:
     <path-segment-reply-list> ::=
                       <path-segment-reply>
                       [<path-segment-reply-list>]

     <path-segment-reply> ::=
                       <CRP>
                       <Tunnel-ID> <Path-ID>
                       <Start-Node>
                       [ <NO-PATH> | <segment-end-List> ]
                       [<metric-list>]
```

6.2.1.5. **Request for Removing Path Segments**

After a shortest path satisfying a set of constraints from source A
to destination Z is computed, a parent controller may delete the path
segments computed and stored in the related child controllers, which
are not any part of the shortest path. A parent controller may send
a child controller a request message for removing the path segments
computed by the child controller and stored in the child controller.

1). A request message for removing path segments (RPSReq message for
short) comprises:

o The global tunnel identifier (GTID).

All the path segments stored under GTID in the child controller are
to be removed.

2). A request message for removing path segments comprises:

o The global tunnel identifier (GTID) and the path identifier (PID).

All the path segments stored under GTID and PID in the child
controller are to be removed.

3). A request message for removing path segments comprises:

o The global tunnel identifier (GTID) and the path identifier (PID)

o A list of start point (or node) addresses or identifiers.

All the path segments stored in the child controller under GTID and
PID and with a start point or node from the list of start point (or
node) addresses or identifiers are to be removed.

4). A request message for removing path segments comprises:

o The global tunnel identifier (GTID) and the path identifier (PID)

o A list of start point (or node) addresses or identifiers

o A list of pairs (start point, a list of end points), which
   identifies the path segments from start point of each pair to each
   of the end points in the list of the pairs.

In addition to the path segments as described in the previous
message, the path segments stored in the child controller under GTID
and PID and identified by the list of pairs are to be removed.

The format of the RPSReq message is as follows:

```
  <RPSReq Message> ::= <Common Header>
                       <remove-path-segment-request-list>
  where:
    <remove-path-segment-request-list> :: =
                       <remove-path-segment-request>
                       [<remove-path-segment-request-list>]

    <remove-path-segment-request> ::=
                       <CRP>
                       <Tunnel-ID> [<Path-ID>]
                       [<start-node-list>]
                       [<branch-List>]

    <start-node-list> ::= <Start-Node> [<start-node-list>]

    <branch-list> ::= <Branch> [<branch-list>]
    <Branch> ::= <Start-Node> <branch-end-list>

    <branch-end-list> ::= <Branch-End> [<branch-end-list>]
```

6.2.1.6. **Reply for Removing Path Segments**

   After removing the path segments as requested by a request message
   for removing path segments from a parent controller, a child
   controller sends the parent controller a reply message for removing
   path segments.

   A reply message for removing path segments (RPSRep message for short)
   comprises:

   o The global tunnel identifier (GTID) and the path identifier (PID)

   o Status of the path segments removal:

      Success: The path segments requested for removal are removed
         successfully.

      Fail: The path segments requested for removal can not be
         removed.

   o Error code and reasons for failure if the status is Fail.

   The format of the RPSRep message is as follows:

```
     <RPSRep Message> ::= <Common Header>
                          <remove-path-segment-reply-list>
     where:
       <remove-path-segment-reply-list> ::=
                          <remove-path-segment-reply>
                          [<remove-path-segment-reply-list>]

       <remove-path-segment-reply> ::=
                          <CRP>
                          <Tunnel-ID> [<Path-ID>]
                          <Status>
                          [<Reasons>]
```

6.2.1.7. **Request for Keeping Path Segments**

   After a shortest path satisfying a set of constraints from source A
   to destination Z is computed, a parent controller may send a request
   message for keeping path segments to each of the related child
   controllers to keep the path segments on the shortest path.

   A request message for keeping path segments (KPSReq message for
   short) comprises:

o The global tunnel identifier (GTID) and the path identifier (PID).

o A list of pairs (start point, end point), each of which identifies
   the path segment from the start point of the pair to the end point
   of the pair.

The child controller will keep the path segments given by the list of
pairs (start point, end point) stored under GTID and PID. It will
remove all the other path segments stored under GTID and PID.

The format of the KPSReq message is as follows:

```
   <KPSReq Message> ::= <Common Header>
                        <keep-path-segment-request-list>
   where:
     <keep-path-segment-request-list> :: =
                        <keep-path-segment-request>
                        [<keep-path-segment-request-list>]

     <keep-path-segment-request> ::=
                        <CRP>
                        <Tunnel-ID> <Path-ID>
                        <segment-list>

     <segment-list> ::= <Segment> [<segment-list>]
     <Segment> ::= <Segment-Start> <Segment-End>
```

6.2.1.8. Reply for Keeping Path Segments

After keeping path segments as requested by a request message for
keeping path segments from a parent controller, a child controller
sends the parent controller a reply message for keeping path
segments.

A reply message for keeping path segments (KPSRep message for short)
comprises:

o The global tunnel identifier (GTID) and the path identifier (PID).

o Status of the path segment retention:

   Success: The path segments requested for retention are retained
     successfully.

Fail: The path segments requested for retention can not be
   retained.

o Error code and reasons for failure if the status is Fail.

The format of the KPSRep message is as follows:

```
<KPSRep Message> ::= <Common Header>
                     <keep-path-segment-reply-list>
where:
  <keep-path-segment-reply-list> ::=
                     <keep-path-segment-reply>
                     [<keep-path-segment-reply-list>]

  <keep-path-segment-reply> ::=
                     <CRP>
                     <Tunnel-ID> <Path-ID>
                     <Status>
                     [<Reasons>]
```

6.2.1.9. Request for Creating Tunnel Segment

After obtaining the end to end shortest point to point (P2P) path, a
parent controller creates a tunnel along the path crossing multiple
domains through sending a request message for creating tunnel segment
to each of the child controllers along the path in a reverse
direction to create a tunnel segment.

A request message for creating tunnel segment (CTSReq message for
short) comprises:

o The global tunnel identifier (GTID) and the path identifier (PID).

o A path segment from a start point to an end point for parent
   without domain topology or a path segment details/ERO for parent
   with domain topology.

o A label and an interface if the domain controlled by the child
   control is not a destination domain.

For parent without domain topology, the child controller allocates
and reserves link bandwidth along the path segment identified by the
start point and end point, assigns labels along the path segment, and
writes cross connects on each of the nodes along the path segment.

For parent with domain topology, the child controller assigns labels
along the path segment ERO and writes cross connects on each of the

nodes along the path segment. The link bandwidth along the path
segment is allocated and reserved by the parent controller.

For the non destination domain, the child controller writes the cross
connect on the edge node to the downstream domain using the label and
the interface from the downstream domain in the message.

For the non source domain, the child controller will include a label
and an interface in a message to be sent to the parent controller.
The interface connects the edge node of the upstream domain along the
path. The label is allocated for the interface on the node that is
the next hop of the edge node.

The format of the CTSReq message is as follows:

```
  <CTSReq Message> ::= <Common Header>
                       <create-tunnel-segment-request-list>
  where:
    <create-tunnel-segment-request-list> ::=
                       <create-tunnel-segment-request>
                       [<create-tunnel-segment-request-list>]

    <create-tunnel-segment-request> ::=
                       <CRP>
                       <Tunnel-ID> <Path-ID>
                       <Path-Segment>
                       [<Label> <Interface>]

    <Path-Segment> ::= [<Segment-Start> <Segment-End> | <ERO> ]
```

6.2.1.10. Reply for Creating Tunnel Segment

After creating tunnel segment as requested by a request message for
creating tunnel segment from a parent controller, a child controller
sends the parent controller a reply message for creating tunnel
segment.

A reply message for creating tunnel segment (CTSRep message for
short) comprises:

o The global tunnel identifier (GTID) and the path identifier (PID).

o Status of the tunnel segment creation:

Success: The tunnel segment requested is created successfully.

Fail: The tunnel segments requested can not be created.

o A label and an interface if the domain controlled by the child
   controller is not source domain and the status is Success.

o Error code and reasons for failure if the status is Fail.

For the non source domain controlled by the child controller, the
interface in the message connects the edge node of the upstream
domain along the path, the label is allocated for the interface on
the node that is the next hop of the edge node.

The format of the CTSRep message is as follows:

```
<CTSRep Message> ::= <Common Header>
                     <create-tunnel-segment-reply-list>
where:
  <create-tunnel-segment-reply-list> ::=
                     <create-tunnel-segment-reply>
                     [<create-tunnel-segment-reply-list>]

  <create-tunnel-segment-reply> ::=
                     <CRP>
                     <Tunnel-ID> <Path-ID>
                     <Status> [<Label> <Interface>]
                     [<Reasons>]
```

6.2.1.11. Request for Removing Tunnel Segment

When a parent controller receives a request for deleting a tunnel
from a user or an application, or receives a reply message for
creating tunnel segment with status of Fail from a child controller,
the parent controller will delete the tunnel through sending a
request message for removing tunnel segment to each of the related
child controllers.

A request message for removing tunnel segment (RTSReq message for
short) comprises:

o The global tunnel identifier (GTID) and the path identifier (PID).

The child controller releases the labels assigned along the path
segments under GTID and PID, and removes the cross connects on each
of the nodes along the path segments. If the child controller
reserved the link bandwidth along the path segments under GTID and

PID, it releases the link bandwidth reserved.

The format of the RTSReq message is as follows:

```
<RTSReq Message> ::= <Common Header>
                     <remove-tunnel-segment-request-list>
where:
  <remove-tunnel-segment-request-list> ::=
                     <remove-tunnel-segment-request>
                     [<remove-tunnel-segment-request-list>]

  <remove-tunnel-segment-request> ::
                     <CRP>
                     <Tunnel-ID> [<Path-ID>]
```

6.2.1.12. Reply for Removing Tunnel Segment

After removing the tunnel segment as requested by a request message
for removing tunnel segment from a parent controller, a child
controller sends the parent controller a reply message for removing
tunnel segment.

A reply message for removing tunnel segment (RTSRep message for
short) comprises:

o The global tunnel identifier (GTID) and the path identifier (PID).

o Status of the tunnel segment removal:

   Success: The tunnel segment requested is removed successfully.

   Fail: The tunnel segment requested can not be removed.

o Error code and reasons for failure if the status is Fail.

The format of the RTSRep message is as follows:

```
     <RTSRep Message> ::= <Common Header>
                          <remove-tunnel-segment-reply-list>
     where:
       <reply-tunnel-segment-reply-list> ::=
                          <remove-tunnel-segment-reply>
                          [<remove-tunnel-segment-reply-list>]

       <remove-tunnel-segment-reply> ::=
                          <CRP>
                          <Tunnel-ID> [<Path-ID>]
                          <Status>
                          [<Reasons>]
```

6.2.2. Individual Encoding of Messages

   The format of PCEP Message Common Header is as follows:

```
     0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     | Ver | Flags |  Message-Type |          Message-Length        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Message-Type (8 bits):   The following message types are currently
      defined (refer to RFC 5440):

```
       Message-Type    Meaning
            1            Open
            2            Keepalive
            3            Path Computation Request
            4            Path Computation Reply
            5            Notification
            6            Error
            7            Close
```

   The new message types are defined as follows:

```
       Message-Type      Meaning
           mTBD1            Controller Relation Discovery
           mTBD2            Connections and Accesses Advertisement
           mTBD3            Path Segment Computation Request
           mTBD4            Path Segment Computation Reply
           mTBD5            Remove Path Segment Request
           mTBD6            Remove Path Segment Reply
           mTBD7            Keep Path Segment Request
           mTBD8            Keep Path Segment Reply
           mTBD9            Create Tunnel Segment Request
           mTBD10           Create Tunnel Segment Reply
           mTBD11           Remove Tunnel Segment Request
           mTBD12           Remove Tunnel Segment Reply
```

Ver, Flags and Message-Length are defined as RFC 5440.

6.2.3. Group Encoding of Messages

We can encode the tunnel related messages into two groups: one group
comprises the request messages related to tunnel and the other
comprises the reply messages related to tunnel. Thus we can have
four new message types, which are defined in PCEP Message Common
Header as follows:

```
       Message-Type      Meaning
           mTBD1            Controller Relation Discovery
           mTBD2            Connections and Accesses Advertisement
           mTBD3            Tunnel Segment Operation Request
           mTBD4            Tunnel Segment Operation Reply
```

Ver, Flags, other message types and Message-Length in PCEP Message
Common Header are defined as RFC 5440.

The Tunnel Segment Operation can be one of the followings:

 Create Tunnel Segment: Create a segment of an end to end tunnel.

 Remove Tunnel Segment: Remove a segment of an end to end tunnel.

 Compute Path Segments: Compute some path segments to find an end to
    end path for an end to end tunnel.

 Remove Path Segments: Remove some path segments.

  Keep Path Segment: Keep path segments on an end to end path for an
     end to end tunnel.

   Each of these operations can be indicated by a value of options field
   of an object such as CRP object following PCEP Message Common Header
   in a message.

6.2.4. Embedded Encoding of Messages

   Each of the request messages can be encoded as a Path Computation
   Request message with a value of options/operations in an existing
   object. Each of the reply messages can be encoded as a Path
   Computation Reply message with a value of options/operations in an
   existing object.

   A new options/operations field of 3 bits may be defined in the
   existing RP object. Thus each of the five request messages for
   supporting HSCS can be represented by a Path Computation Request
   message with a corresponding Options value in the RP object listed
   below. Each of the five reply messages for supporting HSCS can be
   represented by a Path Computation Reply message with a corresponding
   Options value in the RP object listed below.

      Options Value   Meaning
          oTBD1   Path Segment Computation Request/Reply
          oTBD2   Remove Path Segment Request/Reply
          oTBD3   Keep Path Segment Request/Reply
          oTBD4   Create Tunnel Segment Request/Reply
          oTBD5   Remove Tunnel Segment Request/Reply

   Each request/reply message contains the contents for the message
   described in the previous section.

   The Controller Relation Discovery message may be encoded as a Open
   message with a flag or a value of options/operations in an existing
   object. The Open message as a Controller Relation Discovery message
   contains the contents for the Discovery message described in the
   previous section.

   The Connections and Accesses Advertisement message may be encoded as
   a Report message with a flag or a value of options/operations in an
   existing object such as SRP object. The Report message as a
   Connections and Accesses Advertisement message contains the contents
   of the Connections and Accesses Advertisement message described in
   the previous section.

6.2.5. Mixed Encoding of Messages

   Some of the above encodings can be combined to form a mixed encoding
   of the messages for supporting HSCS. For example, one mixed encoding
   of the messages is as follows:

   o Using Individual Encoding for Connections and Accesses
      Advertisement message and

   o Using Embedded Encoding for Controller Relation Discovery, all the
      request and reply messages for supporting HSCS.

   Another mixed encoding of messages is below:

   o Using Embedded Encoding for Controller Relation Discovery;

   o Using Individual Encoding for Connections and Accesses
      Advertisement message and

   o Using Group Encoding for all the request and reply messages for
      supporting HSCS.

6.3. Controller Relation Discovery

   This section presents two approaches for discovering controller
   relation. One uses the Open Message with some simple extensions.
   The other uses a new message for Controller Relation Discovery,
   called a discovery message.

6.3.1. Using Open Message

   For a parent controller P and a child controller C connected by a PCE
   session and having a normal PCE peer adjacency, their parent-child
   relation is discovered through Open Messages exchanged between the
   parent controller and the child controller. The following is a
   sequence of events related to a controller relation discovery.

   Controller P sends controller C an Open Message containing a
   capability TLV with parent flag PC set to 1 after controller C is
   configured as a child controller over the PCE session between P and
   C.

```
                    P                                        C
            Configure C as                          Configure P as
            Child Controller                        Parent Controller

                      Open Message (PC=1)
                    --------------------> Remote P is Parent and
                                          is same as configured
                                          Form Child-Parent relation
                      Open Message (CC=1)
                    <---------------------
            Remote C is Child and
            is same as configured
            Form Parent-Child relation
```

   When C receives the Open Message from P and determines that PC=1 in
   the message is consistent with the parent controller configured
   locally, it forms Child-Parent relation between C and P. It sends
   controller P an Open Message containing a capability TLV with child
   controller flag CC set to 1 after controller P is configured as a
   parent controller over the PCE session between C and P.

   When P receives the Open Message from C and determines that CC=1 in
   the message is consistent with the Child controller configured
   locally, it forms Parent-Child relation between P and C.

   After the Parent-Child relation between P and C is formed, this
   relation is broken if the configuration "C as Child Controller" on
   parent controller P is deleted or "P as Parent Controller" on child
   controller C is removed.

   When the configuration "C as Child Controller" is deleted from parent
   controller P, P breaks/removes the Parent-Child relation between P
   and C and sends C an Open Message with PC = 0. When child controller
   C receives the Open Message with PC = 0 from P, it determines that
   the remote end P is no longer its parent controller as configured
   locally and breaks/removes the Child-Parent relation between C and P.

   When the configuration "P as Parent Controller" is deleted from child
   controller C, C breaks/removes the Child-Parent relation between C
   and P and sends P an Open Message with CC = 0. When parent
   controller P receives the Open Message with CC = 0 from C, it
   determines that the remote end C is no longer its child controller as
   configured locally and breaks/removes the Parent-Child relation
   between P and C.

6.3.2. Using Discovery Message

   For a parent controller P and a child controller C connected by a PCE
   session and having a normal PCE peer adjacency, their parent-child
   relation is discovered through messages for controller relation
   discovery exchanged between the parent controller and the child
   controller. The following is a sequence of events related to a
   controller relation discovery.

   Controller P sends controller C a message containing a local
   controller (LC=) P with a parent flag set to 1 after controller C is
   configured as a child controller over a PCE session between P and C.

```
        P                                     C
 Configure C as child                 Configure P as parent
                 message (LC=P)
              ------------------------> LC in Msg same as configured
                                        Add P as remote controller
               message (LC=C, RC=P)
              <-------------------------
 Remote see me and same as configured
 Form Parent-Child relation
 Add C as remote controller

               message (LC=P, RC=C)
              ------------------------> Remote see me
                                        Form Child-Parent relation
```

   When C receives the message from P and determines that the local
   controller (LC=) P in the message is the same as the parent
   controller configured locally, it sends controller P a message
   containing local controller (LC=) C and remote controller (RC=) P.

   When P receives the message from C and determines that the local
   controller (LC=) C in the message is the same as the child controller
   configured locally and the remote controller C sees me controller P
   (RC=P in the message), it forms a parent-child relation between P and
   C and sends controller C another message containing local controller
   (LC=) P and remote controller (RC=) C.

   When C receives the message from P and determines that the local
   controller (LC=) P in the message is the same as the parent
   controller configured locally and the remote controller P sees me
   controller C (RC=C in the message), it forms a child-parent relation
   between C and P.

6.4. Connections and Accesses Advertisement

A child controller sends its parent controller a message for
connections and accesses, which contains the connections (i.e.,
inter-domain links) connecting the domain that the child controller
controls to other adjacent domains, and the addresses/prefixes (i.e.,
the access points) in the domain to be accessible from outside of the
domain.

When there is a change on the connections and the accesses of the
domain, the child controller sends its parent controller a updated
message for the connections and accesses, which contains the latest
connections and accesses of the domain.

A parent controller stores the connections and accesses for each of
its child controllers according to the messages for connections and
accesses received from the child controllers. For a updated message,
it updates the connections and accesses accordingly.

When a child controller is down, its parent controller may remove the
connections and accesses of the domain controlled by the child
controller.

After connections and accesses advertisement, a parent controller has
the exterior information about all the domains controlled by its
child controllers. In other words, a parent controller has the
connections among the domains (i.e., the inter-domain links
connecting the domains) controlled by its child controllers and the
addresses/prefixes (i.e., access points) in the domains to be
accessible.

A connection comprises: the attributes for a link connecting domains
and the attributes for the end points of the link. The attributes
for an end point of a link comprises the type of the end point node
such as ABR or ASBR, and the domain of the end point such AS number
and area number.

An access point comprises an address or a prefix of a domain to be
accessible outside of the domain.

6.5. Tunnel Creation

This section describes a couple of procedures for computing a
shortest end to end path for a tunnel, and then a procedure for
creating the tunnel along the path. One procedure for computing a
end to end path takes two rounds of computations. The first round
obtains an end to end path without any details on any of the path
segments along the path. This path can be considered as a domain

path. In the second round, the details on each of the path segments
along the domain path are computed. The other procedure is to get an
end to end path in one round.

6.5.1. Computing Path in Two Rounds

After a parent controller receives a request for creating an end to
end tunnel from source A to destination Z for a given set of
constraints, it computes an end to end path in two rounds as follows:

Round 1: Obtain a domain path

>   Roughly speaking, obtaining a domain path consists of the
>   following three steps:

>   Step 1: The parent controller sends a request message to each of
>   its related child controllers for computing a set of path
>   segments in the domain the child controller controls in a
>   special order.

>   Step 2: After a child controller receives the request message, it
>   computes the path segments as requested and sends the parent
>   controller a reply message with the path segments computed as
>   links. It does not store any details about the path segments
>   it computes. The details of the path segments are hidden from
>   the parent controller, which sees each of the segments as a
>   link from one (boundary) node to another (boundary) node with a
>   cost.

>   Step 3: The parent controller builds a shortest path tree (SPT)
>   using these path segments and obtains a shortest path from
>   source A to destination Z that satisfies the constraints.

>   Details for obtaining a domain path are described below:

>   Step 1: The parent controller selects the node just added to the
>   SPT (Initially, it selects the source).

>   Step 2: After selecting the node just added into the SPT, the
>   parent controller chooses the child controller controlling the
>   domain containing the node, and determines whether the node is
>   destination.

>   For destination node, the parent controller stops computing
>   path since the end to end (domain) path from source to
>   destination is in the SPT, which is from the root of the SPT
>   to the node (destination node) in the SPT.

For non-destination node X, the parent controller sends the
   child controller a request message for computing path
   segments in the domain controlled by the child controller.

   o After receiving the request message, the child controller
      computes the path segments as requested and sends the
      parent controller a reply message with the path segments
      computed as links. It does not store any details about
      the path segments it computes. The details of the path
      segments are hidden from the parent controller, which
      sees each of the segments as a link from one (boundary)
      node to another (boundary) node with a cost.

   Step 3: After receiving the reply message from the child
      controller, the parent controller updates the candidate list
      with the links, picks up a node in the candidate list with the
      minimum cost and adds it into the SPT. Repeat step 1.

   Round 2: Obtain the path details

      After obtaining a domain path, the parent controller may
      initiate a BRPC procedure along the domain path to get the end
      to end path. Each of the child controllers controlling the
      domains along the domain path may store the details of the path
      segment it computes using a path key.

6.5.2. Computing Path in One Round

   For a top level parent without domain topology, the parent controller
   computes a shortest point to point (P2P) path for a tunnel from a
   source to a destination satisfying a set of constraints given to the
   tunnel through building a shortest path tree (SPT). The SPT is built
   from the source as the root of the SPT with an empty candidate list
   in the following steps.

   Step 1: The parent controller selects the node just added to the SPT
      (Initially, it selects the source).

   Step 2: After selecting the node just added into the SPT, the parent
      controller chooses the child controller controlling the domain
      containing the node, and determines whether the node is
      destination.

      For destination node, the parent controller stops computing path
         since the end to end path from source to destination is in the
         SPT, which is from the root of the SPT to the node (destination
         node) in the SPT.

For non-destination node X, the parent controller sends the child
   controller a request message for computing path segments
   related to the domain controlled by the child controller. The
   request contains the exception list for the domain and flag E.

   o After receiving the request message, the child controller
      computes a shortest path segment from node X to each of the
      edge nodes of the domain not in the exception list if E is
      1.

   o In addition, it computes a shortest path segment from node X
      to each of the edge nodes of the adjacent domains not in the
      exception list just using the inter-domain links attached to
      node X if node X is an edge node and there is an inter-
      domain link attached to it.

   o If node X is in the destination domain, it computes a
      shortest path segment from node X to the destination.

   o It sends the parent controller a reply message with the path
      segments computed as links and stores the details of the
      path segments temporarily.

Step 3: After receiving the reply message from the child controller,
   the parent controller updates the candidate list with the links,
   picks up a node in the candidate list with the minimum cost and
   adds it into the SPT. Repeat step 1.

For a parent without domain topology, if the parent controller is
also a child controller of another upper level parent controller,
after receiving a request for computing path segments from the upper
level parent controller, the parent controller computes each of the
path segments as requested in the same way as described above. It
records and maintains the path segments computed under the GTID and
PID in the request message received from the upper level parent
controller.

In addition, for each path segment to be computed, it allocates a new
GTID and PID for the path segment and computes the path segment
through sending a request message for computing path segments to each
of its related child controllers using the new GTID and PID.

When the parent as a child controller receives a request message for
removing path segments from the upper level parent controller, it
removes the path segments computed by each of its related child
controllers through sending a request message for removing path
segments to each of the related child controllers, and then it
removes the path segments crossing multiple domains controlled by its

child controllers.

6.5.3. Creating Tunnel along Path

After obtaining the end to end shortest point to point (P2P) path,
the parent controller creates a tunnel along the path crossing
multiple domains through requesting the child controllers along the
path in a reverse direction.

For a parent without domain topology, the following is the procedure
for creating the tunnel along the path, which is initiated by the
parent controller starting from domain X = destination domain.

Step 1: The parent controller sends the child controller controlling
   domain X a request message for creating tunnel segment in domain
   X.

   o   After receiving the request message from the parent controller,
       the child controller creates the tunnel segment in domain X it
       controls through reserving the resources such as link
       bandwidth, allocating labels along the path segment and writing
       a cross connect on every node in the domain along the path.

   o   If the child controller is not destination controller, the
       request message contains an label and interface for the next
       hop of the edge node of domain X. The label is allocated by the
       controller that controls the downstream domain of domain X. The
       child controller uses this label and an incoming label
       allocated for the incoming interface on the edge node to write
       a cross connect on the edge node.

   o   The child controller sends the parent controller a reply
       message with the status of the tunnel segment creation. The
       reply message contains an incoming label and interface for the
       next hop of the edge node of the upstream domain of domain X if
       domain X is not source domain.

Step 2: The parent controller receives the reply message from child
   controller C. If the status in the message is Fail, then it
   removes the tunnel segments created for the tunnel and return with
   failure for creating the tunnel.

Step 3: If child controller C is the source controller, then the end
   to end tunnel is created, and the parent controller and the child
   controllers along the tunnel maintain the information of the
   tunnel with the GTID and PID. The parent controller returns with
   success for creating the tunnel.

Step 4: Child controller C is not source controller. The reply
   message contains the label and interface, the parent controller
   repeats step 1 with domain X = the upstream domain of domain X.
   (In other words, it sends a request message to the child
   controller that controls the domain which is the upstream domain
   of the domain in which a tunnel segment is just created. The
   request contains the label and interface.)

For a parent with domain topology, the procedure for creating the
tunnel along the path initiated by the parent controller is similar
to the one described above, but has a few of changes to it, which are
listed as follows:

o The request message for creating tunnel segment sent to a child
   controller from the parent controller contains the detailed
   information about the path segment (such as ERO comprising every
   hop of the path segment) along which the tunnel segment to be
   created.

o The child controller does not check or reserve resources such as
   link bandwidth along the path segment if the parent controller is
   responsible for allocating and reserving the resources along the
   path for the tunnel.

o The child controller does not assign any labels along the path
   segment if the parent controller is responsible for assigning
   labels along the path for the tunnel. In this case, the request
   message for creating tunnel segment contains an label for every
   hop of the path segment. The reply message from the child
   controller to the parent controller does not contain any label or
   interface.

When the parent as a child controller receives a request message for
creating tunnel segment along a path segment from the upper level
parent controller, it gets the path segments for its related child
controllers from the path segment in the message.

For the parent with domain topology, it obtains the detailed hop to
hop information crossing multiple domains about the path segment
stored by the parent controller using the GTID, PID and start point
and end point of the path segment in the message received. The
parent controller creates the tunnel segments in the multiple domains
through sending a request message for creating tunnel to each of its
related child controllers along the path in a reverse direction.

For the parent without domain topology, it obtains the detailed
information about the path segment stored by the parent controller
using the GTID, PID and start point and end point of the path segment

in the message received. The detailed information includes multiple
path segments, each of which crosses a domain controlled by one of
its related child controllers. These multiple path segments
constitute the path segment in the message, which crosses multiple
domains. The parent controller creates the tunnel segments in the
multiple domains through sending a request message for creating
tunnel to each of its related child controllers along the path in a
reverse direction. For each of the path segments crossing a domain,
the parent controller creates a tunnel segment along the path segment
through sending a request message for creating tunnel to its child
controller controlling the domain.

6.6. Objects and TLVs

6.6.1. CRP Objects

A Controller Request Parameters (CRP) object carried within each of
the new messages for supporting HSCS is used to specify various
parameters of a tunnel related operation request. The CRP object has
Object-Class ocTBD1 and CRP Object-Type = 1. The format of the CRP
body is as follows

```
          Object-Class = ocTBD1 (CRP)    Object-Type = 1
    0                   1         C       2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Flags                                |E|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Request-ID                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                       Optional TLVs                         ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
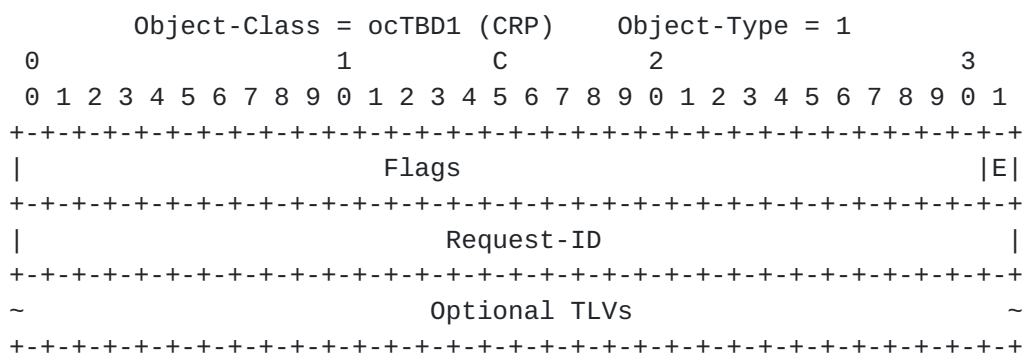
The following flags are currently defined:

o E (Edges of Domain): E set to 1 indicating computing a shortest
   path segment satisfying a given set of constraints from a start
   node to each of the edge nodes of the domain controlled by a child
   controller except for the nodes in a given exception list.

For Group Encoding of messages, a new Options field of 3 bits is
defined in the flags field of the CRP object to tell the receiver of
a message that the request/reply is for one of the five request/reply
messages for supporting HSCS as follows:

```
     Options          Meaning
        1         Path Segment Computation Request/Reply
        2         Remove Path Segment Request/Reply
        3         Keep Path Segment Request/Reply
        4         Create Tunnel Segment Request/Reply
        5         Remove Tunnel Segment Request/Reply
```

6.6.2. LOCAL-CONTROLLER Object

A LOCAL-CONTROLLER (LC) Object is carried within a Controller
Relation Discovery message. Two LC objects are defined: one for IPv4
and the other for IPv6. These two objects have the same Object-Class
ocTBD2 but have different Object-Types.

6.6.2.1. LOCAL-CONTROLLER Object for IPv4

The LOCAL-CONTROLLER Object for IPv4 (LC-IPv4 for short) has Object-
Class ocTBD2 and Object-Type otTBD21. The format of the LC-IPv4 body
is as follows:

```
        Object-Class = ocTBD2     Object-Type = otTBD21
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        Flags                      |P| Level |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                   Controller IPv4 Address                    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ~                       Optional TLVs                          ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The LC-IPv4 object body has a 32-bit Flags field and a 32-bit
Controller IPv4 Address. It may contain additional TLVs. No TLVs
are currently defined.

The following flags are currently defined:

o P (Parent Controller): P set to 1 indicating that the local
   controller is a Parent controller.

o Level (Level as Parent): Level indicates the level of a controller
   as a parent controller. Level 0 means the highest (i.e., top)
   level as a parent controller. Level i (i > 0) for a parent
   controller C means that C as a child controller has a parent
   controller of level (i - 1).

Unassigned bits in the Flags field are considered reserved. They
MUST be set to zero on transmission and MUST be ignored on receipt.

The Controller IPv4 Address indicates an IPv4 address of the local
controller.

6.6.2.2. LOCAL-CONTROLLER Object for IPv6

The LOCAL-CONTROLLER Object for IPv6 (LC-IPv6 for short) has Object-
Class ocTBD2 and Object-Type otTBD22. The format of the LC-IPv6 body
is as follows:

```
        Object-Class = ocTBD2     Object-Type = otTBD22
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Flags                   |P| Level |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                 Controller IPv6 Address               |
 ~                      (16 bytes)                       ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ~                     Optional TLVs                     ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The LC-IPv6 object body has a 32-bit Flags field and a 128-bit
Controller IPv6 Address. It may contain additional TLVs. No TLVs
are currently defined.

The flag P (1 bit) and Level (4 bits) in the 32-bit Flags are the
same as those defined in the LOCAL-CONTROLLER Object for IPv4.

The Controller IPv6 Address indicates an IPv6 address of the local
controller.

6.6.3. REMOTE-CONTROLLER Object

When a local controller receives a Controller Relation Discovery
message from a remote controller, the local controller MUST include a
REMOTE-CONTROLLER (RC) Object with the remote controller in a
Controller Relation Discovery message to be sent to the remote
controller. Two RC objects are defined: one for IPv4 and the other
for IPv6. These two objects have the same Object-Class ocTBD3 but
have different Object-Types.

6.6.3.1. **REMOTE-CONTROLLER Object for IPv4**

   The REMOTE-CONTROLLER Object for IPv4 (RC-IPv4 for short) has Object-
   Class ocTBD3 and Object-Type otTBD31. The format of the RC-IPv4 body
   is as follows:

```
        Object-Class = ocTBD3    Object-Type = otTBD31
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                      Flags                           |P| Level |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                  Controller IPv4 Address                      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ~                       Optional TLVs                          ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
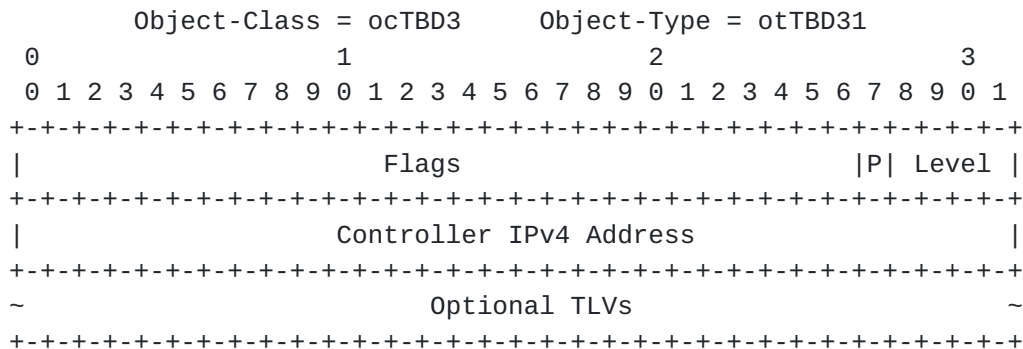
   The RC-IPv4 object body has a 32-bit Flags field and a 32-bit
   Controller IPv4 Address. It may contain additional TLVs. No TLVs
   are currently defined.

   The following flags are currently defined:

   o P (Parent Controller): P set to 1 indicating that the remote
      controller is a Parent controller.

   o Level (Level as Parent): Level indicates the level of a controller
      as a parent controller. Level 0 means the highest (i.e., top)
      level as a parent controller. Level i (i > 0) for a parent
      controller C means that C as a child controller has a parent
      controller of level (i - 1).

   Unassigned bits in the Flags field are considered reserved. They
   MUST be set to zero on transmission and MUST be ignored on receipt.

   The Controller IPv4 Address indicates an IPv4 address of the remote
   controller.

6.6.3.2. **REMOTE-CONTROLLER Object for IPv6**

   The REMOTE-CONTROLLER Object for IPv6 (RC-IPv6 for short) has Object-
   Class ocTBD3 and Object-Type otTBD32. The format of the RC-IPv6 body
   is as follows:

```
           Object-Class = ocTBD3     Object-Type = otTBD32
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                          Flags                       |P| Level |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                    Controller IPv6 Address                    |
    ~                          (16 bytes)                          ~
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    ~                        Optional TLVs                         ~
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The LC-IPv6 object body has a 32-bit Flags field and a 128-bit
Controller IPv6 Address. It may contain additional TLVs. No TLVs
are currently defined.

The flag P (1 bit) and Level (4 bits) in the 32-bit Flags are the
same as those defined in the REMOTE-CONTROLLER Object for IPv4.

The Controller IPv6 Address indicates an IPv6 address of the remote
controller.

6.6.4. CONNECTION and ACCESS Object

The CONNECTION and ACCESS Object (CA for short) has Object-Class
ocTBD4. Three Object-Types are defined under CA object:

   o CA Inter-Domain Link: CA Object-Type is 1.

   o CA Access IPv4 Prefix: CA Object-Type is 2.

   o CA Access IPv6 Prefix: CA Object-Type is 3.

The format of each of these object bodies is as follows:

```
   Object-Class = ocTBD4 (Connection and Access)
   Object-Type = 1 (CA Inter-Domain Link)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          AS Number                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Area-ID TLV                           |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IGP Router-ID TLV                       |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Inter-Domain Link TLVs                    |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Each of the Inter-Domain Link TLVs describes an inter-domain link and
comprises a number of inter-domain link Sub-TLVs.

```
   Object-Class = ocTBD4 (Connection and Access)
   Object-Type = 2 (CA Access 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          AS Number                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Area-ID TLV                           |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Access IPv4 Prefix TLVs                   |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
   Object-Class = ocTBD4 (Connection and Access)
   Object-Type = 3 (CA Access 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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         AS Number                             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         Area-ID TLV                           |
 ~                                                               ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                   Access IPv6 Prefix TLVs                     |
 ~                                                               ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the Area-ID TLV is shown below:

```
  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 (tTBD1)          |           Length (4)         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         Area Number                          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the OSPF Router-ID TLV is shown below:

```
  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 (tTBD2)          |            Length (4)         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         OSPF Router ID                       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the ISIS Router-ID TLV is shown below:

```
  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 (tTBD3)          |            Length (6)         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         ISO Node-ID                          ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the Access IPv4 Prefix TLV is shown as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Type (tTBD4)        |              Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Prefix Length | IPv4 Prefix (variable)                       ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the Access IPv6 Prefix TLV is illustrated below:

```
 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 (tTBD5)        |              Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Prefix Length | IPv6 Prefix (variable)                       ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the Inter-Domain link TLV is illustrated below:

```
 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 (tTBD6)        |              Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Inter-Domain Link Sub-TLVs                   |
~                                                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the Inter-Domain Link Type Sub-TLV is illustrated
below:

```
 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 (1)          |            Length (1)         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Inter-Domain Link Type                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Inter-Domain Link Type sub-TLV defines the type of the inter-
domain link:

   1 - Point-to-point

   2 - Multi-access

   The Inter-Domain Link Type sub-TLV is TLV type 1, and is one octet in
   length.

   The format of the Remote AS Number ID Sub-TLV is illustrated below:

```
   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 (2)           |            Length (4)         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                      Remote AS Number                         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The Remote AS Number field has 4 octets. When only two octets are
   used for the AS number, as in current deployments, the left (high-
   order) two octets MUST be set to zero.

   The format of the Remote Area-ID Sub-TLV is shown below:

```
   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 (3)           |            Length (4)         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                       Area Number                             |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The format of the Remote OSPF Router-ID Sub-TLV is shown below:

```
   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 (4)           |            Length (4)         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                       OSPF Router ID                          |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The format of the Remote ISIS Router-ID Sub-TLV is shown below:

```
    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 (5)        |           Length (6)          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        ISO Node-ID                            ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the IPv4 Remote ASBR ID Sub-TLV is illustrated below:

```
    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 (6)           |             Length (4)        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    IPv4 Remote ASBR ID                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv4 Remote ASBR ID sub-TLV MUST be included if the neighboring
ASBR has an IPv4 address.

The format of the IPv6 Remote ASBR ID Sub-TLV is illustrated below:

```
    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 (7)           |            Length (16)        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    IPv6 Remote ASBR ID                        |
   ~                       (16 Bytes)                              ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv6 Remote ASBR ID sub-TLV MUST be included if the neighboring
ASBR has an IPv6 address.

The format of the Local Interface IPv4 Address Sub-TLV is shown
below:

```
    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 (8)           |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                Local Interface IPv4 Address(es)               |
   ~                                                               ~
```

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Local Interface IPv4 Address sub-TLV specifies the IPv4
address(es) of the interface corresponding to the inter-domain link.
If there are multiple local addresses on the link, they are all
listed in this sub-TLV.

The Local Interface IPv4 Address sub-TLV is TLV type 8, and is 4N
octets in length, where N is the number of local IPv4 addresses.

The format of the Local Interface IPv6 Address Sub-TLV is illustrated
below:

```
  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 (9)          |             Length            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |               Local Interface IPv6 Address(es)                |
 ~                                                               ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Local Interface IPv6 Address sub-TLV specifies the IPv6
address(es) of the interface corresponding to the inter-domain link.
If there are multiple local addresses on the link, they are all
listed in this sub-TLV.

The Local Interface IPv6 Address sub-TLV is TLV type 9, and is 16N
octets in length, where N is the number of local IPv6 addresses.

The format of the Remote Interface IPv4 Address Sub-TLV is
illustrated below:

```
  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 (10)         |             Length            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |              Neighbor Interface IPv4 Address(es)              |
 ~                                                               ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
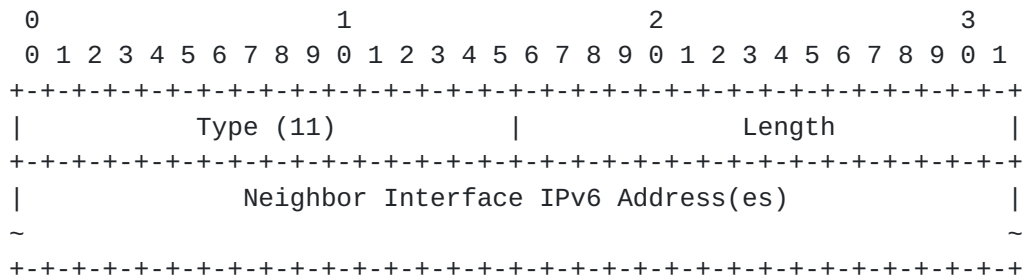
The Remote Interface IPv4 Address sub-TLV specifies the IPv4
address(es) of the neighbor's interface corresponding to the inter-
domain link. This and the local address are used to discern multiple

parallel links between systems. If there are multiple remote
addresses on the link, they are all listed in this sub-TLV.

The Remote Interface IPv4 Address sub-TLV is TLV type 10, and is 4N
octets in length, where N is the number of neighbor IPv4 addresses.

The format of the Remote Interface IPv6 Address Sub-TLV is
illustrated below:

```
 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 (11)          |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Neighbor Interface IPv6 Address(es)               |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Remote Interface IPv6 Address sub-TLV specifies the IPv6
address(es) of the neighbor's interface corresponding to the inter-
domain link. If there are multiple neighbor addresses on the link,
they are all listed in this sub-TLV.

The Remote Interface IPv6 Address sub-TLV is TLV type 11, and is 16N
octets in length, where N is the number of neighbor IPv6 addresses.

6.6.5. NODE Object

The NODE Object has Object-Class ocTBD5. A nuber of Object-Types are
defined under NODE object below:

1. IPv4 START-NODE: NODE Object-Type is 1.

2. IPv6 START-NODE: NODE Object-Type is 2.

3. IPv4 DESTINATION-NODE-LIST: NODE Object-Type is 3.

4. IPv6 DESTINATION-NODE-LIST: NODE Object-Type is 4.

5. IPv4 SEGMENT-END-NODE-LIST: NODE Object-Type is 5.

6. IPv6 SEGMENT-END-NODE-LIST: NODE Object-Type is 6.

7. IPv4 EXCEPTION-NODE-LIST: NODE Object-Type is 7.

   8.   IPv6 EXCEPTION-NODE-LIST: NODE Object-Type is 8.

   9.   NODE-IGP-METRIC-LIST: NODE Object-Type is 9.

   10. NODE-TE-METRIC-LIST: NODE Object-Type is 10.

   11. NODE-HOP-COUNT-LIST: NODE Object-Type is 11.

   The format of NODE object body for IPv4 START-NODE is as follows:

```
    Object-Class = ocTBD5 (NODE)
    Object-Type = 1 (IPv4 START-NODE)
  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
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                    Start Node IPv4 Address                    |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The Start Node IPv4 Address is the IPv4 address of a start node.

   The format of NODE object body for IPv6 START-NODE is as follows:

```
    Object-Class = ocTBD5 (NODE)
    Object-Type = 2 (IPv6 START-NODE)
  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
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                    Start Node IPv6 Address                    |
  ~                         (16 bytes)                            ~
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The Start Node IPv6 Address is the IPv6 address of a start node.

   The format of NODE object body for IPv4 DESTINATION-NODE-LIST is as
   follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 3 (IPv4 DESTINATION-NODE-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Destination Node 1 IPv4 Address              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         . . . . .                            |
~                                                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Destination Node n IPv4 Address              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv4 DESTINATION-NODE-LIST contains n destination node IPv4
addresses. An IPv4 DESTINATION-NODE-LIST is also called an IPv4
DESTINATION-NODES.

The format of NODE object body for IPv6 DESTINATION-NODE-LIST is as
follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 4 (IPv6 DESTINATION-NODE-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Destination Node 1 IPv6 Address              |
~                         (16 bytes)                           ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         . . . . .                            |
~                                                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Destination Node n IPv6 Address              |
~                         (16 bytes)                           ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv6 DESTINATION-NODE-LIST contains n destination node IPv6
addresses. An IPv6 DESTINATION-NODE-LIST is also called an IPv6
DESTINATION-NODES.

The format of NODE object body for IPv4 SEGMENT-END-NODE-LIST is as
follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 5 (IPv4 SEGMENT-END-NODE-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Segment End Node 1 IPv4 Address              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         . . . . .                            |
~                                                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Segment End Node n IPv4 Address              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv4 SEGMENT-END-NODE-LIST contains n segment node IPv4
addresses. An IPv4 SEGMENT-END-NODE-LIST is also called an IPv4
SEGMENT-END-NODES.

The format of NODE object body for IPv6 SEGMENT-END-NODE-LIST is as
follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 6 (IPv6 SEGMENT-END-NODE-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Segment End Node 1 IPv6 Address              |
~                         (16 bytes)                           ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         . . . . .                            |
~                                                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Segment End Node n IPv6 Address              |
~                         (16 bytes)                           ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv6 SEGMENT-END-NODE-LIST contains n segment end node IPv6
addresses. An IPv6 SEGMENT-END-NODE-LIST is also called an IPv6
SEGMENT-END-NODES.

The format of NODE object body for IPv4 EXCEPTION-NODE-LIST is as
follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 7 (IPv4 EXCEPTION-NODE-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Exception Node 1 IPv4 Address                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       .  .  .  .  .  .                        |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Exception Node n IPv4 Address                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv4 SEGMENT-END-NODE-LIST contains n node IPv4 addresses in an
exception list. An IPv4 EXCEPTION-NODE-LIST is also called an IPv4
EXCEPTION-LIST.

The format of NODE object body for IPv6 EXCEPTION-NODE-LIST is as
follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 8 (IPv6 EXCEPTION-NODE-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Exception Node 1 IPv6 Address                 |
~                         (16 bytes)                            ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       .  .  .  .  .  .                        |
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Exception Node n IPv6 Address                 |
~                         (16 bytes)                            ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The IPv6 EXCEPTION-NODE-LIST contains n node IPv6 addresses in an
exception list. An IPv6 EXCEPTION-NODE-LIST is also called an IPv6
EXCEPTION-LIST.

The format of NODE object body for NODE-IGP-METRIC-LIST is as
follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 9 (NODE-IGP-METRIC-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Segment End Node 1 IGP Metric Value             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         .  .  .  .  .                        |
~                                                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Segment End Node n IGP Metric Value             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The NODE-IGP-METRIC-LIST contains n IGP metrics for n segment end
nodes.

The format of NODE object body for NODE-TE-METRIC-LIST is as follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 10 (NODE-TE-METRIC-LIST)
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Segment End Node 1 TE Metric Value              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         .  .  .  .  .                        |
~                                                              ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Segment End Node n TE Metric Value              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The NODE-TE-METRIC-LIST contains n TE metrics for n segment end
nodes.

The format of NODE object body for NODE-HOP-COUNT-LIST is as follows:

```
   Object-Class = ocTBD5 (NODE)
   Object-Type = 11 (NODE-HOP-COUNT-LIST)
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Segment End Node 1 Hop Counts Value               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        . . . . . .                           |
 ~                                                              ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Segment End Node n Hop Counts Value               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The NODE-HOP-COUNT-LIST contains n hop counts values for n segment
end nodes.

6.6.6. TUNNEL Object

The TUNNEL Object has Object-Class ocTBD6. Two Object-Types are
defined under TUNNEL object:

 1. TUNNEL-ID: TUNNEL Object-Type is 1.

 2. TUNNEL-PATH-ID: TUNNEL Object-Type is 2.

The format of TUNNEL object body for TUNNEL-ID is as follows:

```
   Object-Class = ocTBD6 (TUNNEL)
   Object-Type = 1 (TUNNEL-ID)
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         Tunnel ID                            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Tunnel ID in the body is a 32-bit unique number for identifying a
tunnel globally.

The format of TUNNEL object body for TUNNEL-PATH-ID is as follows:

```
   Object-Class = ocTBD6 (TUNNEL)   Object-Type = 2 (PATH-ID)
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          Path ID                             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Path ID in the body is a 16-bit number for uniquely identifying a
path under a tunnel.

6.6.7. STATUS Object

The STATUS Object has Object-Class ocTBD7. The format of STATUS
object body has following format:

```
   Object-Class = ocTBD7 (STATUS)
   Object-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
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Status Code |    Reason     |            Reserved           |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  ~                      Optional TLVs                          ~
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The status code (or status for short) in a STATUS may be one of the
followings:

 1 (SUCCESS): Indicating a request is successfully finished.

 2 (FAIL): Indicating a request can not be finished.

When the status is FAIL, the Reason gives a reason for the failure
and the Optional TLVs give some more details about failure.

6.6.8. LABEL Object

The LABEL Object has Object-Class ocTBD8. The format of LABEL object
body has following format:

```
   Object-Class = ocTBD8 (LABEL)
   Object-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.

6.6.9. INTERFACE Object

The INTERFACE Object has Object-Class ocTBD9. Three Object-Types are
defined under INTERFACE object:

 1.   Index: Object-Type is 1.

 2.   IPv4 Address: Object-Type is 2.

 3.   IPv6 Address: Object-Type is 3.

The format of INTERFACE object body for interface index has following
format:

```
    Object-Class = ocTBD9 (INTERFACE)
    Object-Type = 1 (Index)
 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
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                       Interface Index                         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Interface Index is a single interface index, encoded in 4 octets.

The format of INTERFACE object body for interface IPv4 address has
following format:

```
    Object-Class = ocTBD9 (INTERFACE)
    Object-Type = 2 (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
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                    Interface IPv4 Address                     |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Interface IPv4 Address is a single interface IPv4 address,
encoded in 4 octets.

The format of INTERFACE object body for interface IPv6 address has
following format:

```
      Object-Class = ocTBD9 (INTERFACE)
      Object-Type = 3 (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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                    Interface IPv6 Address                     |
    ~                         (16 bytes)                            ~
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Interface IPv6 Address is a single interface IPv6 address,
encoded in 16 octets.

7. Security Considerations

The mechanism described in this document does not raise any new
security issues for the PCEP protocols.

8. IANA Considerations

This section specifies requests for IANA allocation.

9. Acknowledgement

The authors would like to thank people for their valuable comments on
this draft.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
           RFC2119, March 1997,
           <http://www.rfc-editor.org/info/rfc2119>.

[RFC4655] Farrel, A., Vasseur, J., and J. Ash, "A Path Computation
           Element (PCE)-Based Architecture", RFC 4655, DOI 10.17487/
           RFC4655, August 2006,
           <http://www.rfc-editor.org/info/rfc4655>.

[RFC5440] Vasseur, JP., Ed. and JL. Le Roux, Ed., "Path Computation
           Element (PCE) Communication Protocol (PCEP)", RFC 5440,
           DOI 10.17487/RFC5440, March 2009,
           <http://www.rfc-editor.org/info/rfc5440>.

[RFC5441] Vasseur, JP., Ed., Zhang, R., Bitar, N., and JL. Le Roux,
           "A Backward-Recursive PCE-Based Computation (BRPC)
```

              Procedure to Compute Shortest Constrained Inter-Domain
              Traffic Engineering Label Switched Paths", RFC 5441,
              DOI 10.17487/RFC5441, April 2009,
              <http://www.rfc-editor.org/info/rfc5441>.

   [RFC5392]  Chen, M., Zhang, R., and X. Duan, "OSPF Extensions in
              Support of Inter-Autonomous System (AS) MPLS and GMPLS
              Traffic Engineering", RFC 5392, DOI 10.17487/RFC5392,
              January 2009, <http://www.rfc-editor.org/info/rfc5392>.

   [RFC5316]  Chen, M., Zhang, R., and X. Duan, "ISIS Extensions in
              Support of Inter-Autonomous System (AS) MPLS and GMPLS
              Traffic Engineering", RFC 5316, DOI 10.17487/RFC5316,
              December 2008, <http://www.rfc-editor.org/info/rfc5316>.

   [RFC5305]  Li, T. and H. Smit, "IS-IS Extensions for Traffic
              Engineering", RFC 5305, DOI 10.17487/RFC5305,
              October 2008, <http://www.rfc-editor.org/info/rfc5305>.

   [RFC3630]  Katz, D., Kompella, K., and D. Yeung, "Traffic Engineering
              (TE) Extensions to OSPF Version 2", RFC 3630,
              DOI 10.17487/RFC3630, September 2003,
              <http://www.rfc-editor.org/info/rfc3630>.

## 10.2.  Informative References

   [RFC1136]  Hares, S. and D. Katz, "Administrative Domains and Routing
              Domains: A model for routing in the Internet", RFC 1136,
              DOI 10.17487/RFC1136, December 1989,
              <http://www.rfc-editor.org/info/rfc1136>.

   [RFC4105]  Le Roux, J., Ed., Vasseur, J., Ed., and J. Boyle, Ed.,
              "Requirements for Inter-Area MPLS Traffic Engineering",
              RFC 4105, DOI 10.17487/RFC4105, June 2005,
              <http://www.rfc-editor.org/info/rfc4105>.

   [RFC4216]  Zhang, R., Ed. and J. Vasseur, Ed., "MPLS Inter-Autonomous
              System (AS) Traffic Engineering (TE) Requirements",
              RFC 4216, DOI 10.17487/RFC4216, November 2005,
              <http://www.rfc-editor.org/info/rfc4216>.

   [RFC6006]  Zhao, Q., Ed., King, D., Ed., Verhaeghe, F., Takeda, T.,
              Ali, Z., and J. Meuric, "Extensions to the Path
              Computation Element Communication Protocol (PCEP) for
              Point-to-Multipoint Traffic Engineering Label Switched
              Paths", RFC 6006, DOI 10.17487/RFC6006, September 2010,
              <http://www.rfc-editor.org/info/rfc6006>.

   [RFC6805]  King, D., Ed. and A. Farrel, Ed., "The Application of the
              Path Computation Element Architecture to the Determination
              of a Sequence of Domains in MPLS and GMPLS", RFC 6805,
              DOI 10.17487/RFC6805, November 2012,
              <http://www.rfc-editor.org/info/rfc6805>.

## Appendix A.  Details on Embedded Encoding of Messages

   A new options field of 3 bits is defined in the flags field of the RP
   object to tell the receiver of the message that the request/reply is
   for one of the five request/reply messages for supporting HSCS as
   follows:

       Options           Meaning
          1        Path Segment Computation Request/Reply
          2        Remove Path Segment Request/Reply
          3        Keep Path Segment Request/Reply
          4        Create Tunnel Segment Request/Reply
          5        Remove Tunnel Segment Request/Reply


   A new flag E of 1 bit is defined in the flags field of the RP object.
   Flag E set to 1 indicating computing a shortest path segment
   satisfying a given set of constraints from a start node to each of
   the edge nodes of the domain controlled by a child controller except
   for the nodes in a given exception list.

## A.1.  Message for Controller Relation Discovery

   The new TLV defined in the Open Object in section Capability
   Discovery is extended to contain Sub-TLVs for local controller and
   remote controller.  Thus Open Message with the Open Object containing
   the new TLV can be used as Message for Controller Relation Discovery.
   Four optional Sub-TLVs are defined as follows:

   1.  Local Controller IPv4 Sub-TLV

       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 (1)          |            Length             |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                     Flags                     |P| Level |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                   Controller IPv4 Address                     |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       ~                   (Optional Sub-TLVs)                         ~
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

   The meanings of each field in the Sub-TLV is the same as described in
   section LOCAL-CONTROLLER Object for IPv4.

   2.  Local Controller IPv6 Sub-TLV

```
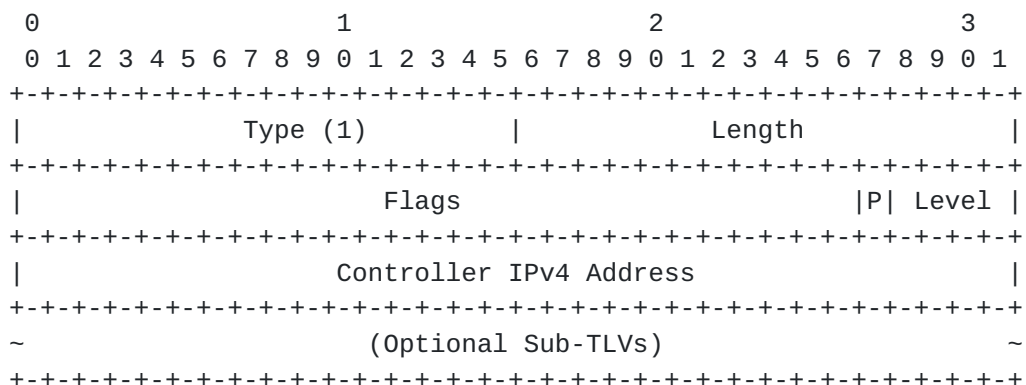    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |             Type (2)          |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Flags                         |P| Level |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  Controller IPv6 Address                      |
   ~                         (16 bytes)                            ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                     (Optional Sub-TLVs)                       ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The meanings of each field in the Sub-TLV is the same as described in
   section LOCAL-CONTROLLER Object for IPv6.

   3. Remote Controller IPv4 Sub-TLV

```
    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 (3)          |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Flags                         |P| Level |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  Controller IPv4 Address                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                     (Optional Sub-TLVs)                       ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The meanings of each field in the Sub-TLV is the same as described in
   section REMOTE-CONTROLLER Object for IPv4.

   4. Remote Controller IPv6 Sub-TLV

```
 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 (4)         |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Flags                            |P| Level |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Controller IPv6 Address                       |
~                        (16 bytes)                             ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                    (Optional Sub-TLVs)                        ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The meanings of each field in the Sub-TLV is the same as described in
section REMOTE-CONTROLLER Object for IPv6.

A.2. Message for Connections and Accesses Advertisement

The format of the CAAdv message is as follows:

```
<CAAdv Message> ::= <Common Header>
                    <SRP>
                    <Inter-Domain-Link-List>
                    [<Access-Address-List>]
where:
<Inter-Domain-Link-List> ::= <Inter-Domain-Link>
                             [<Inter-Domain-Link-List>]
<Access-Address-List> ::= <Access-Address>
                          [<Access-Address-List>]
```

A.3. Request for Computing Path Segments

The format of the PSReq message is as follows:

```
   <PSReq Message> ::= <Common Header>
                       [<svec-list>]
                       <path-segment-request-list>
   where:
     <svec-list>::=<SVEC>[<svec-list>]
     <path-segment-request-list> ::=
                       <path-segment-request>
                       [<path-segment-request-list>]

     <path-segment-request> ::=
                       <RP> <END-POINTS> [<OF>] [<LSPA>] [<BANDWIDTH>]
                       <Tunnel-ID> <Path-ID>
                       [<metric-list>] [<RRO>[<BANDWIDTH>]] [<IRO>]
                       [<LOAD-BALANCING>]
                       <exception-list>
```

[A.4](#). Reply for Computing Path Segments

   The format of the PSRep message is as follows:

```
   <PSRep Message> ::= <Common Header>
                       <path-segment-reply-list>
   where:
     <path-segment-reply-list> ::=
                       <path-segment-reply>
                       [<path-segment-reply-list>]

     <path-segment-reply> ::=
                       <RP> [<NO-PATH>] [<attribute-list>]
                       <Tunnel-ID> <Path-ID>
                       <Start-Node>
                       [ <NO-PATH> | <segment-end-List> ]
                       [<attribute-list>]
```

[A.5](#). Request for Removing Path Segments

   The format of the RPSReq message is as follows:

```
   <RPSReq Message> ::= <Common Header>
                        <remove-path-segment-request-list>
   where:
     <remove-path-segment-request-list> :: =
                        <remove-path-segment-request>
                        [<remove-path-segment-request-list>]

     <remove-path-segment-request> ::=
                        <RP>
                        <Tunnel-ID> [<Path-ID>]
                        [<start-node-list>]
                        [<branch-List>]

     <start-node-list> ::= <Start-Node> [<start-node-list>]

     <branch-list> ::= <Branch> [<branch-list>]
     <Branch> ::= <Start-Node> <branch-end-list>

     <branch-end-list> ::= <Branch-End> [<branch-end-list>]
```

A.6. Reply for Removing Path Segments

   The format of the RPSRep message is as follows:

```
   <RPSRep Message> ::= <Common Header>
                        <remove-path-segment-reply-list>
   where:
     <remove-path-segment-reply-list> ::=
                        <remove-path-segment-reply>
                        [<remove-path-segment-reply-list>]

     <remove-path-segment-reply> ::=
                        <RP>
                        <Tunnel-ID> [<Path-ID>]
                        <Status>
                        [<Reasons>]
```

A.7. Request for Keeping Path Segments

   The format of the KPSReq message is as follows:

```
      <KPSReq Message> ::= <Common Header>
                           <keep-path-segment-request-list>
      where:
        <keep-path-segment-request-list> :: =
                           <keep-path-segment-request>
                           [<keep-path-segment-request-list>]

        <keep-path-segment-request> ::=
                           <RP>
                           <Tunnel-ID> <Path-ID>
                           <segment-list>

        <segment-list> ::= <Segment> [<segment-list>]
        <Segment> ::= <Segment-Start> <Segment-End>
```

A.8. Reply for Keeping Path Segments

   The format of the KPSRep message is as follows:

```
      <KPSRep Message> ::= <Common Header>
                           <keep-path-segment-reply-list>
      where:
        <keep-path-segment-reply-list> ::=
                           <keep-path-segment-reply>
                           [<keep-path-segment-reply-list>]

        <keep-path-segment-reply> ::=
                           <RP>
                           <Tunnel-ID> <Path-ID>
                           <Status>
                           [<Reasons>]
```

A.9. Request for Creating Tunnel Segment

   The format of the CTSReq message is as follows:

```
<CTSReq Message> ::= <Common Header>
                     <create-tunnel-segment-request-list>
where:
  <create-tunnel-segment-request-list> ::=
                     <create-tunnel-segment-request>
                     [<create-tunnel-segment-request-list>]

  <create-tunnel-segment-request> ::=
                     <RP>
                     <Tunnel-ID> <Path-ID>
                     <Path-Segment>
                     [<Label> <Interface>]

  <Path-Segment> ::= [<Segment-Start> <Segment-End> | <ERO> ]
```

A.10. Reply for Creating Tunnel Segment

The format of the CTSRep message is as follows:

```
<CTSRep Message> ::= <Common Header>
                     <create-tunnel-segment-reply-list>
where:
  <create-tunnel-segment-reply-list> ::=
                     <create-tunnel-segment-reply>
                     [<create-tunnel-segment-reply-list>]

  <create-tunnel-segment-reply> ::=
                     <RP>
                     <Tunnel-ID> <Path-ID>
                     <Status> [<Label> <Interface>]
                     [<Reasons>]
```

A.11. Request for Removing Tunnel Segment

The format of the RTSReq message is as follows:

```
<RTSReq Message> ::= <Common Header>
                     <remove-tunnel-segment-request-list>
where:
  <remove-tunnel-segment-request-list> ::=
                     <remove-tunnel-segment-request>
                     [<remove-tunnel-segment-request-list>]

  <remove-tunnel-segment-request> ::
                     <RP>
                     <Tunnel-ID> [<Path-ID>]
```

A.12. **Reply for Removing Tunnel Segment**

   The format of the RTSRep message is as follows:

      <RTSRep Message> ::= <Common Header>
                          <remove-tunnel-segment-reply-list>
      where:
        <reply-tunnel-segment-reply-list> ::=
                          <remove-tunnel-segment-reply>
                          [<remove-tunnel-segment-reply-list>]

        <remove-tunnel-segment-reply> ::=
                          <RP>
                          <Tunnel-ID> [<Path-ID>]
                          <Status>
                          [<Reasons>]

Authors' Addresses

   Huaimo Chen
   Huawei Technologies
   Boston, MA,
   USA

   EMail: Huaimo.chen@huawei.com

   Mehmet Toy
   USA

   EMail: mtoy054@yahoo.com

   Lei Liu
   Fujitsu
   USA

   EMail: lliu@us.fujitsu.com

      Vic Liu
      China Mobile
      No.32 Xuanwumen West Street, Xicheng District
      Beijing,   100053
      China

      EMail: liuzhiheng@chinamobile.com