MPLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 2, 2018

S. Bryant, Ed.
Huawei
A. Farrel, Ed.
J. Drake
Juniper Networks
J. Tantsura
Individual

August 29, 2017

MPLS Segment Routing in IP Networks
draft-bryant-mpls-unified-ip-sr-02

Abstract

Segment routing is a source routed forwarding method that allows packets to be steered through a network on paths other than the shortest path derived from the routing protocol. The approach uses information encoded in the packet header to partially or completely specify the route the packet takes through the network, and does not make use of a signaling protocol to pre-install paths in the network.

Two different encapsulations have been defined to enable segment routing in an MPLS network or in an IPv6 network. While acknowledging that there is a strong need to support segment routing in both environments, this document defines a mechanism to carry MPLS segment routing packets encapsulated in UDP. The resulting approach is applicable to both IPv4 and IPv6 networks without the need for any changes to the IP or segment routing specifications.

This document makes no changes to the segment routing architecture and builds on existing protocol mechanisms such as the encapsulation of MPLS within UDP defined in RFC 7510.

No new procedures are introduced, but existing mechanisms are combined to achieve the desired result.

Requirements Language

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].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Segment routing (SR) [I-D.ietf-spring-segment-routing] is a source routed forwarding method that allows packets to be steered through a network on paths other than the shortest path derived from the routing protocol. SR also allows the packets to be steered through a set of packet processing functions along that path. SR uses information encoded in the packet header to partially or completely specify the route the packet takes through the network and does not make use of a signaling protocol to pre-install paths in the network.

The approach to segment routing in IPv6 networks is known as SRv6 and is described in [I-D.ietf-6man-segment-routing-header]. The mechanism described encodes the segment routing instruction list as an ordered list of 128-bit IPv6 addresses that is carried in a new IPv6 extension header: the Source Routing Header (SRH).

MPLS-SPRING [I-D.ietf-spring-segment-routing-mpls] (also known as MPLS Segment Routing or MPLS-SR) encodes the route the packet takes through the network and the instructions to be applied to the packet as it transits the network by imposing a stack of MPLS label entries on the packet.

This document describes a method for running SR in IPv4 or IPv6 networks by using an MPLS-SR label stack carried in UDP. No change is made to the MPLS-SR encoding mechanism as described in [I-D.ietf-spring-segment-routing-mpls] where a sequence of 32 bit units, one for each instruction, called the Segment Routing Instruction Stack (SRIS) is used. Each basic unit is encoded as an MPLS label stack entry and the segment routing instructions (i.e., the Segment Identifiers, SIDs) are encoded in the 20 bit MPLS Label fields.

In summary, the processing described in this document is a combination of normal MPLS-over-UDP behavior as described in [RFC7510], MPLS-SR lookup and label-pop behavior as described in [I-D.ietf-spring-segment-routing-mpls], and normal IP forwarding. No new procedures are introduced, but existing mechanisms are combined to achieve the desired result.

The method defined is a complementary way of running SR in an IP network that can be used alongside or interchangeably with that defined in [I-D.ietf-6man-segment-routing-header]. Implementers and deployers should consider the benefits and drawbacks of each method and select the approach most suited to their needs.

2. The MPLS-SR-over-UDP Encoding Stack

The MPLS-SR-over-UDP encoding stack is shown in Figure 1.

```
    +---------------------+
    |                     |
    |      IP Header      |
    |                     |
    +---------------------+
    |                     |
    |     UDP Header      |
    |                     |
    +---------------------+
    |                     |
    |   Segment Routing   |
    |  Instruction Stack  |
    ~                     ~
    ~                     ~
    |                     |
    +---------------------+
    |                     |
    |      Payload        |
    ~                     ~
    ~                     ~
    |                     |
    +---------------------+
```

Figure 1: Packet Encapsulation

The payload may be of any type that, with an appropriate convergence layer, can be carried over a packet network. It is anticipated that the most common packet types will be IPv4, IPv6, native MPLS, and pseudowires [RFC3985].

Preceding the Payload is the Segment Routing Instruction Stack (SRIS) that carries the sequence of instructions to be executed on the packet as it traverses the network. This is the Segment Identifier (SID) stack that is the ordered list of segments described in [I-D.ietf-spring-segment-routing].

Preceding the SRIS is a UDP header. The UDP header is included to:

- o Introduce entropy to allow equal-cost multi-path load balancing (ECMP) [RFC2992] in the IP layer [RFC7510].
- o Provide a protocol multiplexing layer as an alternative to using a new IP type/next header.
- o Allow transit through firewalls and other middleboxes.
- o Provide disaggregation.

Preceding the UDP header is the IP header which may be IPv4 or IPv6.

3. The Segment Routing Instruction Stack

The SRIS consists of a sequence of Segment Identifiers as described in [I-D.ietf-spring-segment-routing] encoded as an MPLS label stack as described in [I-D.ietf-spring-segment-routing-mpls].

The top SRIS entry is the next instruction to be executed. When the node to which this instruction is directed has processed the instruction it is removed (popped) from the SRIS, and the next instruction processed.

Each instruction is encoded in a single Label Stack Entry (LSE) as shown in Figure 2. The structure of the LSE is unchanged from [RFC3032].

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Instruction                | TC  |S|     TTL     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                Instruction:  Label Value, 20 bits
                TC:           Traffic Class, 3 bits
                S:            Bottom of Stack, 1 bit
                TTL:          Time to Live, 8 bits
```

Figure 2: SRIS Label Stack Entry

As with [I-D.ietf-spring-segment-routing-mpls] a 32 bit LSE is used to carry each SR instruction. The instruction itself is carried in the 20 bit Label Value field. The TC field has the normal meaning as defined in [RFC3032] and modified in [RFC5462]. The S bit has bottom of stack semantics defined in [RFC3032]. TTL is discussed in Section 3.1.

3.1. TTL

The setting of the TTL is application specific, but the following operational consideration should be born in mind. In SR the size of the label stack may be increased within a single routing domain by various operations such as the pushing of a binding SID. Furthermore in SR packets are not necessarily constrained to travel on the shortest path with that routing domain. Consideration therefore has to be given to possibility of a forwarding loop. To mitigate against this it is RECOMMENDED that the TTL is continuously decremented as the packet passes through the SR network regardless of any other changes to the network layer encapsulation.

Further discussion of the use of TTL during tunnelling can be found in [RFC4023].

4. UDP/IP Encapsulation

The procedures defined in [RFC7510] are followed. RFC7510 specifies the values to be used in the UDP Source Port, Destination Port, and Checksum fields.

An administrative domain, or set of administrative domains that are sufficiently well managed and monitored to be able to safely use IP segment routing is likely to comply with the requirements called out in [RFC7510] to permit operation with a zero checksum over IPv6. However each operator needs to validate the decision on whether or not to use a UDP checksum for themselves.

The [RFC7510] UDP header may be carried over IPv4 or over IPv6.

The IP source address is the address of the encapsulating device. The IP destination address is implied by the instruction at the top of the instruction stack.

If IPv4 is in use, fragmentation is not permitted.

5. Elements of Procedure

Not all of the nodes in an SR domain are "SR capable" meaning that they can process MPLS-SR packets. Some nodes may be "legacy routers" that cannot handle SR packets but can forward IP packets. An SR capable node may advertise its capabilities using the IGP as described in Section 7. There are six types of node in an SR domain:

- o Domain ingress nodes that receive packets and encapsulate them for transmission across the domain. These packets may be any payload protocol including native IP packets or packets that are already MPLS encapsulated.

- o Legacy transit nodes that are IP routers but that are not able to perform segment routing.

- o Transit nodes that are SR capable but that are not identified by a SID in the SID stack.

- o Transit nodes that are SR capable and need to perform SR routing.

- o The penultimate SR capable node on the path that processes the last SID on the stack on behalf of the domain egress node.

- o The domain egress node that forwards the payload packet for ultimate delivery.

The following sub-sections describe the processing behavior in each case.

In summary, the processing is a combination of normal MPLS-over-UDP behavior as described in [RFC7510], MPLS-SR lookup and label-pop behavior as described in [I-D.ietf-spring-segment-routing-mpls], and normal IP forwarding. No new procedures are introduced, but existing mechanisms ae combined to achieve the desired result.

The descriptions in the following sections represent the functional behavior. Optimizations on this behavior may be possible in implementations.

5.1. Domain Ingress Nodes

Domain ingress nodes receive packets from outside the domain and encapsulate them to be forwarded across the domain. Received packets may already be MPLS-SR packets (in the case of connecting two MPLS-SR networks across a native IP network), or may be IP or MPLS packets.

In the latter case, the packet is classified by the domain ingress node and an MPLS-SR stack is imposed. In the former case the MPLS-SR stack is already in the packet. The top entry in the stack is popped from the stack and retained for use below.

The packet is then encapsulated in UDP with the destination port set to 6635 to indicate "MPLS-UDP" or to 6636 to indicate "MPLS-UDP-DTLS"as described in [RFC7510]. The source UDP port is set randomly or to provide entropy as described in [RFC7510].

The packet is then encapsulated in IP for transmission across the network. The IP source address is set to the domain ingress node, and the destination address is set to the address corresponding to the label that was previously popped from the stack.

This corresponds to sending the packet out of a virtual interface that corresponds to a virtual link between the ingress node and the next hop SR node realized by a UDP tunnel.

The packet is then sent into the IP network and is routed according to the local FIB and applying hashing to resolve any ECMP choices.

5.2. Legacy Transit Nodes

A legacy transit node is an IP router that has no SR capabilities. When such a router receives an MPLS-SR-in-UDP packet it will carry out normal TTL processing and if the packet is still live it will forward it as it would any other UDP-in-IP packet. The packet will be routed toward the destination indicated in the packet header using the local FIB and applying hashing to resolve any ECMP choices.

If the packet is mistakenly addressed to the legacy router, the UDP tunnel will be terminated and the packet will be discarded either because the MPLS-in-UDP port is not supported or because the uncovered top label has not been allocated. This is, however, a misconnection and should not occur unless there is a routing error.

5.3. On-Path Pass-Through SR Nodes

Just because a node is SR capable and receives an MPLS-SR-in-UDP packet does not mean that it performs SR processing on the packet. Only routers identified by SIDs in the SR stack need to do such processing.

Routers that are not addressed by the destination address in the IP header simply treat the packet as a normal UDP-in-IP packet carrying out normal TTL processing and if the packet is still live routing the packet according to the local FIB and applying hashing to resolve any ECMP choices.

This is important because it means that the SR stack can be kept relatively small and the packet can be steered through the network using shortest path first routing between selected SR nodes.

5.4. SR Transit Nodes

An SR capable node that is addressed by the top most SID in the stack when that is not the last SID in the stack (i.e., the S bit is not set) is an SR transit node. When an SR transit node receives an MPLS-SR-in-UDP packet that is addressed to it, it acts as follows:

- o Perform TTL processing as normal for an IP packet.
- o Determine that the packet is addressed to the local node.
- o Find that the payload is UDP and that the destination port indicates MPLS-in-UDP.
- o Strip the IP and UDP headers.
- o Pop the top label from the SID stack and retain it for use below.
- o Encapsulate the packet in UDP with the destination port set to 6635 (or 6636 for DTLS) and the source port set for entropy. The entropy value SHOULD be retained from the received UDP header or MAY be freshly generated since this is a new UDP tunnel.
- o Encapsulate the packet in IP with the IP source address set to this transit router, and the destination address set to the address corresponding to the next SID in the stack.
- o Send the packet into the IP network routing the packet according to the local FIB and applying hashing to resolve any ECMP choices.

5.5. Penultimate SR Transit Nodes

The penultimate SR transit node is an SR transit node as described in Section 5.4 where the SID for the node is directly followed by the final SID (i.e., that of domain egress node). When a penultimate SR transit node receives an MPLS-SR-in-UDP packet that is addressed to it, it acts according to whether penultimate hop popping (PHP) is supported for the final SID. That information could be indicated using the control plane as described in Section 7.

If PHP is allowed the penultimate SR transit node acts as follows:

- o Perform TTL processing as normal for an IP packet.
- o Determine that the packet is addressed to the local node.
- o Find that the payload is UDP and that the destination port indicates MPLS-in-UDP.

- Strip the IP and UDP headers.

- Pop the top label from the SID stack and retain it for use below.

- Pop the next label from the SID stack.

- Encapsulate the packet in UDP with the destination port set to 6635 (or 6636 for DTLS) and the source port set for entropy. The entropy value SHOULD be retained from the received UDP header or MAY be freshly generated since this is a new UDP tunnel.

- Encapsulate the packet in IP with the IP source address set to this transit router, and the destination address set to the domain egress node IP address corresponding to the label that was previously popped from the stack.

- Send the packet into the IP network routing the packet according to the local FIB and applying hashing to resolve any ECMP choices.

If PHP is not supported, the penultimate SR transit node just acts as a normal SR transit node just as described in Section 5.4. However, the penultimate SR transit node may be required to replace the final SID with an MPLS-SR label stack entry carrying an explicit null label value (0 for IPv4 and 2 for IPv6) before forwarding the packet. This requirement may also be indicated by the control plane as described in Section 7.

5.6. Domain Egress Nodes

The domain egress acts as follows:

- Perform TTL processing as normal for an IP packet.

- Determine that the packet is addressed to the local node.

- Find that the payload is UDP and that the destination port indicates MPLS-in-UDP.

- Strip the IP and UDP headers.

- Pop the outermost SID if present (i.e., if PHP was not performed as described in Section 5.5.

- Pop the explicit null label if it is present in the label stack as requested by the domain egress and communicated in the control plane as described in Section 7.

   o Forward the payload packet according to its type and the local
      routing/forwarding mechanisms.

6. Modes of Deployment

As previously noted, the procedures described in this document may be used to connect islands of SR functionality across an IP backbone, or can provide SR function within a native IP network. This section briefly expounds upon those two deployment modes.

6.1. Interconnection of SR Domains

Figure 3 shows two SR domains interconnected by an IP network. The procedures described in this document are deployed at border routers R1 and R2 and packets are carried across the backbone network in a UDP tunnel.

R1 acts as the domain ingress as described in Section 5.1. It takes the MPLS-SR packet from the SR domain, pops the top label and uses it to identify its peer border router R2. R1 then encapsulates the packet in UDP in IP and sends it toward R2.

Routers within the IP network simply forward the packet using normal IP routing.

R2 acts as a domain egress router as described in Section 5.6. It receives a packet that is addressed to it, strips the IP and UDP headers, and acts on the payload SR label stack to continue to route the packet.

```
                    ________________________
       ______      (                        )      ______
      (      )    (        IP Network        )    (      )
     (        )  (                            )  (        )
    (      --------                          --------      )
   (      | Border |    SR-in-UDP Tunnel    | Border |      )
   (  SR  | Router |========================| Router |  SR  )
   (      |   R1   |                        |   R2   |      )
    (      --------                          --------      )
     (        )  (                            )  (        )
      (______)    (                          )    (______)
                   (________________________)
```

Figure 3: SR in UDP to Tunnel Between SR Sites

6.2. SR Within an IP Network

Figure 4 shows the procedures defined in this document to provide SR function across an IP network.

R1 receives a native packet and classifies it, determining that it should be sent on the SR path R2-R3-R4-R5. It imposes a label stack accordingly and then acts as a domain ingress as described in Section 5.1. It pops the label for R2, and encapsulates the packet in UDP in IP, sets the IP source to R1 and the IP destination to R2, and sends the packet into the IP network.

Routers Ra and Rb are transit routers that simply forward the packets using normal IP forwarding. They may be legacy transit routers (see Section 5.2) or on-path pass-through SR nodes (see Section 5.3).

R2 is an SR transit nodes as described in Section 5.4. It receives a packet addressed to it, strips the IP and UDP headers, and processes the SR label stack. It pops the top label and uses it to identify the next SR hop which is R3. R2 then encapsulates the packet in UDP in IP setting the IP source to R2 and the IP destination to R3.

Rc, Rd, and Re are transit routers and perform as Ra and Rb.

R3 is an SR transit node and performs as R2.

R4 is a penultimate SR transit node as described in Section 5.5. It receives a packet addressed to it, strips the IP and UDP headers, and processes the SR label stack. It pops the top label and uses it to identify the next SR hop which is R5.

R5 is the domain egress as described in Section 5.6. It receives a packet addressed to it, strips the IP and UDP headers.

```
                    __________________________________
                 __(           IP Network             )__
              __(                                        )__
             (               --        --        --         )
        --------   --   --  |R2|  --  |R3|  --  |R4|  --   --------
       | Ingress| |Ra| |Rb| |  | |Rc| |  | |Rd| |  | |Re| | Egress |
   --->| Router |===========|  | |======|  |======|  |======| Router |--->
       |   R1   | |  | |  | |  | |  | |  | |  | |  | |  | |   R5   |
        --------   --   --  |  |  --  |  |  --  |  |  --   --------
             (__             --        --        --       __)
                (__                                    __)
                   (__________________________________)
```

Figure 4: SR Within an IP Network

7. Control Plane

This document is concerned with forwarding plane issues, and a description of applicable control plane mechanisms is out of scope. This section is provided only as a collection of references. No changes to the control plane mechanisms for MPLS-SR are needed or proposed.

A routers that is able to support SR can advertise the fact in the IGP as follows:

- o In IS-IS, by using the SR-Capabilities TLV as defined in [I-D.ietf-isis-segment-routing-extensions]

- o In OSPF/OSPFv3 by using the Router Information LSA as defined in [I-D.ietf-ospf-segment-routing-extensions] and [I-D.ietf-ospf-ospfv3-segment-routing-extensions].

Nodes can advertise SIDs using the mechanisms defined in [I-D.ietf-isis-segment-routing-extensions], [I-D.ietf-ospf-segment-routing-extensions], or [I-D.ietf-ospf-ospfv3-segment-routing-extensions].

Support for PHP can be indicated in a SID advertisement using flags in the advertisements as follows:

- o For IS-IS, the N (no-PHP) flag in the Prefix-SID sub-TLV indicates whether PHP is not to be used.

- o For OSPF/OSPFv3, the NP (no-PHP) flag in the Prefix SID Sub-TLV indicates whether PHP is not to be used.

The requirement to use an explicit null SID if PHP is not in use can be indicated in SID advertisement using the Explicit-Null Flag (E-Flag). If set, the penultimate SR transit node replaces the final SID with a SID containing an Explicit-NULL value (0 for IPv4 and 2 for IPv6) before forwarding the packet.

The method of advertising the tunnel encapsulation capability of a router using IS-IS or OSPF are specified in [I-D.ietf-isis-encapsulation-cap] and [I-D.ietf-ospf-encapsulation-cap] respectively. No changes to those procedures are needed in support of this work.

8. OAM

OAM at the payload layer follows the normal OAM procedures for the payload. To the payload the whole SR network looks like a tunnel.

OAM in the IP domain follows the normal IP procedures. This can only be carried out between on the IP hops between pairs of SR nodes.

OAM between instruction processing entities i.e. at the SR layer uses the procedures documented for MPLS.

9. Security Considerations

The security consideration of [I-D.ietf-spring-ipv6-use-cases] and [RFC7510] apply. DTLS [RFC6347] SHOULD be used where security is needed on an MPLS-SR-over-UDP segment.

It is difficult for an attacker to pass a raw MPLS encoded packet into a network and operators have considerable experience at excluding such packets at the network boundaries.

It is easy for an ingress node to detect any attempt to smuggle IP packet into the network since it would see that the UDP destination port was set to MPLS. SR packets not having a destination address terminating in the network would be transparently carried and would pose no security risk to the network under consideration.

10. IANA Considerations

This document makes no IANA requests.

11. Acknowledgements

This draft was partly inspired by [I-D.xu-mpls-unified-source-routing-instruction], and we acknowledge the following authors of version -02 of that draft: Robert Raszuk, Uma Chunduri, Luis M. Contreras, Luay Jalil, Hamid Assarpour, Gunter Van De Velde, Jeff Tantsura, and Shaowen Ma.

Thanks to Joel Halpern, Bruno Decraene, Loa Andersson, Ron Bonica, Eric Rosen, Robert Raszuk, Wim Henderickx, Jim Guichard, and Gunter Van De Velde for their insightful comments on this draft.

12. Contributors

- o Mach Chen, Huawei Technologies, mach.chen@huawei.com

13. References

13.1. Normative References

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-12 (work in progress), June 2017.

[I-D.ietf-spring-segment-routing-mpls]
Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with MPLS data plane", draft-ietf-spring-segment-routing-mpls-10 (work in progress), June 2017.

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

[RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001, <https://www.rfc-editor.org/info/rfc3032>.

[RFC5462] Andersson, L. and R. Asati, "Multiprotocol Label Switching (MPLS) Label Stack Entry: "EXP" Field Renamed to "Traffic Class" Field", RFC 5462, DOI 10.17487/RFC5462, February 2009, <https://www.rfc-editor.org/info/rfc5462>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black, "Encapsulating MPLS in UDP", RFC 7510, DOI 10.17487/RFC7510, April 2015, <https://www.rfc-editor.org/info/rfc7510>.

13.2. Informative References

[I-D.ietf-6man-segment-routing-header]
Previdi, S., Filsfils, C., Raza, K., Leddy, J., Field, B., daniel.voyer@bell.ca, d., daniel.bernier@bell.ca, d., Matsushima, S., Leung, I., Linkova, J., Aries, E., Kosugi, T., Vyncke, E., Lebrun, D., Steinberg, D., and R. Raszuk, "IPv6 Segment Routing Header (SRH)", draft-ietf-6man-segment-routing-header-07 (work in progress), July 2017.

[I-D.ietf-isis-encapsulation-cap]
Xu, X., Decraene, B., Raszuk, R., Chunduri, U., Contreras, L., and L. Jalil, "Advertising Tunnelling Capability in IS-IS", draft-ietf-isis-encapsulation-cap-01 (work in progress), April 2017.

[I-D.ietf-isis-segment-routing-extensions]
Previdi, S., Filsfils, C., Bashandy, A., Gredler, H., Litkowski, S., Decraene, B., and j. jefftant@gmail.com, "IS-IS Extensions for Segment Routing", draft-ietf-isis-segment-routing-extensions-13 (work in progress), June 2017.

[I-D.ietf-ospf-encapsulation-cap]
Xu, X., Decraene, B., Raszuk, R., Contreras, L., and L. Jalil, "Advertising Tunneling Capability in OSPF", draft-ietf-ospf-encapsulation-cap-06 (work in progress), July 2017.

[I-D.ietf-ospf-ospfv3-segment-routing-extensions]
Psenak, P., Previdi, S., Filsfils, C., Gredler, H., Shakir, R., Henderickx, W., and J. Tantsura, "OSPFv3 Extensions for Segment Routing", draft-ietf-ospf-ospfv3-segment-routing-extensions-09 (work in progress), March 2017.

[I-D.ietf-ospf-segment-routing-extensions]
Psenak, P., Previdi, S., Filsfils, C., Gredler, H., Shakir, R., Henderickx, W., and J. Tantsura, "OSPF Extensions for Segment Routing", draft-ietf-ospf-segment-routing-extensions-19 (work in progress), August 2017.

[I-D.ietf-spring-ipv6-use-cases]
Brzozowski, J., Leddy, J., Filsfils, C., Maglione, R., and M. Townsley, "IPv6 SPRING Use Cases", draft-ietf-spring-ipv6-use-cases-11 (work in progress), June 2017.

[I-D.xu-mpls-unified-source-routing-instruction]
Xu, X., Filsfils, C., Bashandy, A., Raszuk, R., Chunduri, U., Contreras, L., Jalil, L., Assarpour, H., Velde, G., Tantsura, J., Ma, S., and T. Mizrahi, "Unified Source Routing Instructions using MPLS Label Stack", draft-xu-mpls-unified-source-routing-instruction-03 (work in progress), August 2017.

[RFC2992] Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, DOI 10.17487/RFC2992, November 2000, <https://www.rfc-editor.org/info/rfc2992>.

[RFC3985] Bryant, S., Ed. and P. Pate, Ed., "Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture", RFC 3985, DOI 10.17487/RFC3985, March 2005, <https://www.rfc-editor.org/info/rfc3985>.

[RFC4023] Worster, T., Rekhter, Y., and E. Rosen, Ed., "Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)", RFC 4023, DOI 10.17487/RFC4023, March 2005, <https://www.rfc-editor.org/info/rfc4023>.

Authors' Addresses

Stewart Bryant (editor)
Huawei

Email: stewart.bryant@gmail.com

Adrian Farrel (editor)
Juniper Networks

Email: afarrel@juniper.net

John Drake
Juniper Networks

Email: jdrake@juniper.net

Jeff Tantsura
Individual

Email: jefftant.ietf@gmail.com

MPLS Working Group   S. Bryant, Ed.
Internet-Draft   Huawei
Intended status: Standards Track   A. Farrel, Ed.
Expires: May 3, 2018   J. Drake
                                                        Juniper Networks
                                                             J. Tantsura
                                                              Individual
                                                        October 30, 2017

MPLS Segment Routing in IP Networks
draft-bryant-mpls-unified-ip-sr-03

Abstract

Segment routing is a source routed forwarding method that allows packets to be steered through a network on paths other than the shortest path derived from the routing protocol. The approach uses information encoded in the packet header to partially or completely specify the route the packet takes through the network, and does not make use of a signaling protocol to pre-install paths in the network.

Two different encapsulations have been defined to enable segment routing in an MPLS network or in an IPv6 network. While acknowledging that there is a strong need to support segment routing in both environments, this document defines a mechanism to carry MPLS segment routing packets encapsulated in UDP. The resulting approach is applicable to both IPv4 and IPv6 networks without the need for any changes to the IP or segment routing specifications.

This document makes no changes to the segment routing architecture and builds on existing protocol mechanisms such as the encapsulation of MPLS within UDP defined in RFC 7510.

No new procedures are introduced, but existing mechanisms are combined to achieve the desired result.

Requirements Language

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].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Segment routing (SR) [I-D.ietf-spring-segment-routing] is a source routed forwarding method that allows packets to be steered through a network on paths other than the shortest path derived from the routing protocol. SR also allows the packets to be steered through a set of packet processing functions along that path. SR uses information encoded in the packet header to partially or completely specify the route the packet takes through the network and does not make use of a signaling protocol to pre-install paths in the network.

The approach to segment routing in IPv6 networks is known as SRv6 and is described in [I-D.ietf-6man-segment-routing-header]. The mechanism described encodes the segment routing instruction list as an ordered list of 128-bit IPv6 addresses that is carried in a new IPv6 extension header: the Source Routing Header (SRH).

MPLS Segment Routing (MPLS-SR) [I-D.ietf-spring-segment-routing-mpls] encodes the route the packet takes through the network and the instructions to be applied to the packet as it transits the network by imposing a stack of MPLS label stack entries on the packet.

This document describes a method for running SR in IPv4 or IPv6 networks by using an MPLS-SR label stack carried in UDP. No change is made to the MPLS-SR encoding mechanism as described in [I-D.ietf-spring-segment-routing-mpls] where a sequence of 32 bit units, one for each instruction, called the Segment Routing Instruction Stack (SRIS) is used. Each basic unit is encoded as an MPLS label stack entry and the segment routing instructions (i.e., the Segment Identifiers, SIDs) are encoded in the 20 bit MPLS Label fields.

In summary, the processing described in this document is a combination of normal MPLS-over-UDP behavior as described in [RFC7510], MPLS-SR lookup and label-pop behavior as described in [I-D.ietf-spring-segment-routing-mpls], and normal IP forwarding. No new procedures are introduced, but existing mechanisms are combined to achieve the desired result.

The method defined is a complementary way of running SR in an IP network that can be used alongside or interchangeably with that defined in [I-D.ietf-6man-segment-routing-header]. Implementers and deployers should consider the benefits and drawbacks of each method and select the approach most suited to their needs.

2. The MPLS-SR-over-UDP Encoding Stack

The MPLS-SR-over-UDP encoding stack is shown in Figure 1.

```
    +---------------------+
    |                     |
    |      IP Header      |
    |                     |
    +---------------------+
    |                     |
    |     UDP Header      |
    |                     |
    +---------------------+
    |                     |
    |   Segment Routing   |
    |  Instruction Stack  |
    ~                     ~
    ~                     ~
    |                     |
    +---------------------+
    |                     |
    |       Payload       |
    ~                     ~
    ~                     ~
    |                     |
    +---------------------+
```

Figure 1: Packet Encapsulation

The payload may be of any type that, with an appropriate convergence layer, can be carried over a packet network. It is anticipated that the most common packet types will be IPv4, IPv6, native MPLS, and pseudowires [RFC3985].

Preceding the Payload is the Segment Routing Instruction Stack (SRIS) that carries the sequence of instructions to be executed on the packet as it traverses the network. This is the Segment Identifier (SID) stack that is the ordered list of segments described in [I-D.ietf-spring-segment-routing].

Preceding the SRIS is a UDP header. The UDP header is included to:

o Introduce entropy to allow equal-cost multi-path load balancing (ECMP) [RFC2992] in the IP layer [RFC7510].

o Provide a protocol multiplexing layer as an alternative to using a new IP type/next header.

o Allow transit through firewalls and other middleboxes.

o Provide disaggregation.

Preceding the UDP header is the IP header which may be IPv4 or IPv6.

3. The Segment Routing Instruction Stack

The Segment Routing Instruction Stack (SRIS) consists of a sequence of Segment Identifiers (SIDs) as described in [I-D.ietf-spring-segment-routing] encoded as an MPLS label stack as described in [I-D.ietf-spring-segment-routing-mpls].

The top SRIS entry is the next instruction to be executed. When the node to which this instruction is directed has processed the instruction it is removed (popped) from the SRIS, and the next instruction is processed.

Each instruction is encoded in a single Label Stack Entry (LSE) as shown in Figure 2. The structure of the LSE is unchanged from [RFC3032].

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Instruction                  | TC  |S|   TTL     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                Instruction:  Label Value, 20 bits
                TC:           Traffic Class, 3 bits
                S:            Bottom of Stack, 1 bit
                TTL:          Time to Live, 8 bits
```

Figure 2: SRIS Label Stack Entry

As with [I-D.ietf-spring-segment-routing-mpls] a 32 bit LSE is used to carry each SR instruction. The instruction itself is carried in the 20 bit Label Value field. The TC field has the normal meaning as defined in [RFC3032] and modified in [RFC5462]. The S bit has bottom of stack semantics defined in [RFC3032]. TTL is discussed in Section 3.1.

3.1. TTL

The setting of the TTL is application specific, but the following operational consideration should be born in mind. In SR the size of the label stack may be increased within a single routing domain by various operations such as the pushing of a Binding SID. Furthermore, in SR packets are not necessarily constrained to travel on the shortest path within a routing domain. Therefore, considration has to be given to the possibility that there may be a forwarding loop. To mitigate against this it is RECOMMENDED that the TTL is decremented at each hop as the packet passes through the SR network regardless of any other changes to the network layer encapsulation.

Further discussion of the use of TTL during tunnelling can be found in [RFC4023].

4. UDP/IP Encapsulation

[RFC7510] specifies the values to be used in the UDP Source Port, Destination Port, and Checksum fields.

An administrative domain, or set of administrative domains that are sufficiently well managed and monitored to be able to safely use IP segment routing is likely to comply with the requirements called out in [RFC7510] to permit operation with a zero UDP checksum over IP. However each operator needs to validate the decision on whether or not to use a UDP checksum for themselves.

The [RFC7510] UDP header may be carried over IPv4 or over IPv6.

The IP source address is the address of the encapsulating device. The IP destination address is implied by the instruction at the top of the instruction stack.

If IPv4 is in use, fragmentation is not permitted.

5. Elements of Procedure

Nodes that are SR capable can process MPLS-SR packets. Not all of the nodes in an SR domain are SR capable. Some nodes may be "legacy routers" that cannot handle SR packets but can forward IP packets. An SR capable node may advertise its capabilities using the IGP as described in Section 8. There are six types of node in an SR domain:

- o Domain ingress nodes that receive packets and encapsulate them for transmission across the domain. Those packets may be any payload protocol including native IP packets or packets that are already MPLS encapsulated.

- o Legacy transit nodes that are IP routers but that are not SR capable (i.e., are not able to perform segment routing).

- o Transit nodes that are SR capable but that are not identified by a SID in the SID stack.

- o Transit nodes that are SR capable and need to perform SR routing because they are identified by a SID in the SID stack.

- o The penultimate SR capable node on the path that processes the last SID on the stack on behalf of the domain egress node.

- o The domain egress node that forwards the payload packet for ultimate delivery.

The following sub-sections describe the processing behavior in each case.

In summary, the processing is a combination of normal MPLS-over-UDP behavior as described in [RFC7510], MPLS-SR lookup and label-pop behavior as described in [I-D.ietf-spring-segment-routing-mpls], and normal IP forwarding. No new procedures are introduced, but existing mechanisms ae combined to achieve the desired result.

The descriptions in the following sections represent the functional behavior. Optimizations on this behavior may be possible in implementations.

5.1. Domain Ingress Nodes

Domain ingress nodes receive packets from outside the domain and encapsulate them to be forwarded across the domain. Received packets may already be MPLS-SR packets (in the case of connecting two MPLS-SR networks across a native IP network), or may be nativee IP or MPLS packets.

In the latter case, the packet is classified by the domain ingress node and an MPLS-SR stack is imposed. In the former case the MPLS-SR stack is already in the packet. The top entry in the stack is popped from the stack and retained for use below.

The packet is then encapsulated in UDP with the destination port set to 6635 to indicate "MPLS-UDP" or to 6636 to indicate "MPLS-UDP-DTLS" as described in [RFC7510]. The source UDP port is set randomly or to provide entropy as described in [RFC7510].

The packet is then encapsulated in IP for transmission across the network. The IP source address is set to the domain ingress node, and the destination address is set to the address corresponding to the label that was previously popped from the stack.

This processing is equivalent to sending the packet out of a virtual interface that corresponds to a virtual link between the ingress node and the next hop SR node realized by a UDP tunnel.

The packet is then sent into the IP network and is routed according to the local FIB and applying hashing to resolve any ECMP choices.

5.2. Legacy Transit Nodes

A legacy transit node is an IP router that has no SR capabilities. When such a router receives an MPLS-SR-in-UDP packet it will carry out normal TTL processing and if the packet is still live it will forward it as it would any other UDP-in-IP packet. The packet will be routed toward the destination indicated in the packet header using the local FIB and applying hashing to resolve any ECMP choices.

If the packet is mistakenly addressed to the legacy router, the UDP tunnel will be terminated and the packet will be discarded either because the MPLS-in-UDP port is not supported or because the uncovered top label has not been allocated. This is, however, a misconnection and should not occur unless there is a routing error.

5.3. On-Path Pass-Through SR Nodes

Just because a node is SR capable and receives an MPLS-SR-in-UDP packet does not mean that it performs SR processing on the packet. Only routers identified by SIDs in the SR stack need to do such processing.

Routers that are not addressed by the destination address in the IP header simply treat the packet as a normal UDP-in-IP packet carrying out normal TTL processing and if the packet is still live routing the packet according to the local FIB and applying hashing to resolve any ECMP choices.

This is important because it means that the SR stack can be kept relatively small and the packet can be steered through the network using shortest path first routing between selected SR nodes.

5.4. SR Transit Nodes

An SR capable node that is addressed by the top most SID in the stack when that is not the last SID in the stack (i.e., the S bit is not set) is an SR transit node. When an SR transit node receives an MPLS-SR-in-UDP packet that is addressed to it, it acts as follows:

- o Perform TTL processing as normal for an IP packet.
- o Determine that the packet is addressed to the local node.
- o Find that the payload is UDP and that the destination port indicates MPLS-in-UDP.
- o Strip the IP and UDP headers.
- o Pop the top label from the SID stack and retain it for use below.
- o Encapsulate the packet in UDP with the destination port set to 6635 (or 6636 for DTLS) and the source port set for entropy. The entropy value SHOULD be retained from the received UDP header or MAY be freshly generated since this is a new UDP tunnel.
- o Encapsulate the packet in IP with the IP source address set to this transit router, and the destination address set to the address corresponding to the next SID in the stack.
- o Send the packet into the IP network routing the packet according to the local FIB and applying hashing to resolve any ECMP choices.

5.5. Penultimate SR Transit Nodes

The penultimate SR transit node is an SR transit node as described in Section 5.4 where the SID for the node is directly followed by the final SID (i.e., that of domain egress node). When a penultimate SR transit node receives an MPLS-SR-in-UDP packet that is addressed to it, it acts according to whether penultimate hop popping (PHP) is supported for the final SID. That information could be indicated using the control plane as described in Section 8. It is worth making some additional observations about PHP in SR: these are collected in Section 6.

If PHP is allowed the penultimate SR transit node acts as follows:

- o Perform TTL processing as normal for an IP packet.
- o Determine that the packet is addressed to the local node.

- o Find that the payload is UDP and that the destination port indicates MPLS-in-UDP.

- o Strip the IP and UDP headers.

- o Pop the top label from the SID stack and retain it for use below.

- o Pop the next label from the SID stack.

- o Encapsulate the packet in UDP with the destination port set to 6635 (or 6636 for DTLS) and the source port set for entropy. The entropy value SHOULD be retained from the received UDP header or MAY be freshly generated since this is a new UDP tunnel.

- o Encapsulate the packet in IP with the IP source address set to this transit router, and the destination address set to the domain egress node IP address corresponding to the label that was previously popped from the stack.

- o Send the packet into the IP network routing the packet according to the local FIB and applying hashing to resolve any ECMP choices.

If PHP is not supported, the penultimate SR transit node just acts as a normal SR transit node just as described in Section 5.4. However, the penultimate SR transit node may be required to replace the final SID with an MPLS-SR label stack entry carrying an explicit null label value (0 for IPv4 and 2 for IPv6) before forwarding the packet. This requirement may also be indicated by the control plane as described in Section 8.

5.6. Domain Egress Nodes

The domain egress acts as follows:

- o Perform TTL processing as normal for an IP packet.

- o Determine that the packet is addressed to the local node.

- o Find that the payload is UDP and that the destination port indicates MPLS-in-UDP.

- o Strip the IP and UDP headers.

- o Pop the outermost SID if present (i.e., if PHP was not performed as described in Section 5.5.

- o Pop the explicit null label if it is present in the label stack as requested by the domain egress and communicated in the control plane as described in Section 8.

- o Forward the payload packet according to its type and the local routing/forwarding mechanisms.

6. A Note on Segment Routing Paths and Penultimate Hop Popping

End-to-end SR paths are comprised of multiple segments. The end point of each segment is identified by a SID in the SID stack.

In normal SR processing a penultimate hop is the router that performs SR routing immediately prior to the end of segment router. Penultimate hop popping (PHP) is processing that applies at the penultimate router in a segment.

With MPLS-SR-in-UDP encapsulation, each SR segment is achieved using using an MPLS-in-UDP tunnel that runs the full length of the segment. The SR SID stack on a packet is only examined at the head and tail of this segment. Thus, each segment is effectively one hop long in the SR overlay network and if there is any PHP processing it takes place at the head-end of the segment.

However, in order to simplify processing at each MPLS-SR-in-UDP end point, it is RECOMMENDED that PHP processing is only used for the final segment in an SR path as described in Section 5.5.

7. Modes of Deployment

As previously noted, the procedures described in this document may be used to connect islands of SR functionality across an IP backbone, or can provide SR function within a native IP network. This section briefly expounds upon those two deployment modes.

7.1. Interconnection of SR Domains

Figure 3 shows two SR domains interconnected by an IP network. The procedures described in this document are deployed at border routers R1 and R2 and packets are carried across the backbone network in a UDP tunnel.

R1 acts as the domain ingress as described in Section 5.1. It takes the MPLS-SR packet from the SR domain, pops the top label and uses it to identify its peer border router R2. R1 then encapsulates the packet in UDP in IP and sends it toward R2.

Routers within the IP network simply forward the packet using normal IP routing.

R2 acts as a domain egress router as described in Section 5.6. It receives a packet that is addressed to it, strips the IP and UDP headers, and acts on the payload SR label stack to continue to route the packet.

```
                    ________________________
       ______      (                        )      ______
      (      )    (        IP Network        )    (      )
     (        )  (                            )  (        )
    (      --------                          --------      )
   (      | Border |    SR-in-UDP Tunnel    | Border |      )
   (  SR  | Router |========================| Router |  SR  )
   (      |   R1   |                        |   R2   |      )
    (      --------                          --------      )
     (        )  (                            )  (        )
      (______)    (                          )    (______)
                   (________________________)
```

Figure 3: SR in UDP to Tunnel Between SR Sites

7.2. SR Within an IP Network

Figure 4 shows the procedures defined in this document to provide SR function across an IP network.

R1 receives a native packet and classifies it, determining that it should be sent on the SR path R2-R3-R4-R5. It imposes a label stack accordingly and then acts as a domain ingress as described in Section 5.1. It pops the label for R2, and encapsulates the packet in UDP in IP, sets the IP source to R1 and the IP destination to R2, and sends the packet into the IP network.

Routers Ra and Rb are transit routers that simply forward the packets using normal IP forwarding. They may be legacy transit routers (see Section 5.2) or on-path pass-through SR nodes (see Section 5.3).

R2 is an SR transit nodes as described in Section 5.4. It receives a packet addressed to it, strips the IP and UDP headers, and processes the SR label stack. It pops the top label and uses it to identify the next SR hop which is R3. R2 then encapsulates the packet in UDP in IP setting the IP source to R2 and the IP destination to R3.

Rc, Rd, and Re are transit routers and perform as Ra and Rb.

R3 is an SR transit node and performs as R2.

R4 is a penultimate SR transit node as described in Section 5.5. It receives a packet addressed to it, strips the IP and UDP headers, and processes the SR label stack. It pops the top label and uses it to identify the next SR hop which is R5.

R5 is the domain egress as described in Section 5.6. It receives a packet addressed to it, strips the IP and UDP headers.

```
                    __________________________________
                 __(           IP Network             )__
              __(                                        )__
             (               --        --        --         )
        --------   --   --  |R2|  --  |R3|  --  |R4|  --   --------
       | Ingress| |Ra| |Rb|  |  | |Rc| |  | |Rd| |  | |Re| | Egress |
   --->| Router |===========|  |======|  |======|  |======| Router |--->
       |   R1   | |  | |  | |  | |  | |  | |  | |  | |  | |   R5   |
        --------   --   --  |  |  --  |  |  --  |  |  --   --------
             (__             --        --        --       __)
                (__                                    __)
                   (__________________________________)
```

Figure 4: SR Within an IP Network

8. Control Plane

This document is concerned with forwarding plane issues, and a description of applicable control plane mechanisms is out of scope. This section is provided only as a collection of references. No changes to the control plane mechanisms for MPLS-SR are needed or proposed.

A routers that is able to support SR can advertise the fact in the IGP as follows:

o In IS-IS, by using the SR-Capabilities TLV as defined in [I-D.ietf-isis-segment-routing-extensions]

o In OSPF/OSPFv3 by using the Router Information LSA as defined in [I-D.ietf-ospf-segment-routing-extensions] and [I-D.ietf-ospf-ospfv3-segment-routing-extensions].

Nodes can advertise SIDs using the mechanisms defined in [I-D.ietf-isis-segment-routing-extensions], [I-D.ietf-ospf-segment-routing-extensions], or [I-D.ietf-ospf-ospfv3-segment-routing-extensions].

Support for PHP can be indicated in a SID advertisement using flags in the advertisements as follows:

- o For IS-IS, the N (no-PHP) flag in the Prefix-SID sub-TLV indicates whether PHP is not to be used.

- o For OSPF/OSPFv3, the NP (no-PHP) flag in the Prefix SID Sub-TLV indicates whether PHP is not to be used.

The requirement to use an explicit null SID if PHP is not in use can be indicated in SID advertisement using the Explicit-Null Flag (E-Flag). If set, the penultimate SR transit node replaces the final SID with a SID containing an Explicit-NULL value (0 for IPv4 and 2 for IPv6) before forwarding the packet.

The method of advertising the tunnel encapsulation capability of a router using IS-IS or OSPF are specified in [I-D.ietf-isis-encapsulation-cap] and [I-D.ietf-ospf-encapsulation-cap] respectively. No changes to those procedures are needed in support of this work.

9. OAM

OAM at the payload layer follows the normal OAM procedures for the payload. To the payload the whole SR network looks like a tunnel.

OAM in the IP domain follows the normal IP procedures. This can only be carried out between on the IP hops between pairs of SR nodes.

OAM between instruction processing entities i.e., at the SR layer uses the procedures documented for MPLS.

10. Security Considerations

The security consideration of [I-D.ietf-spring-ipv6-use-cases] and [RFC7510] apply. DTLS [RFC6347] SHOULD be used where security is needed on an MPLS-SR-over-UDP segment.

It is difficult for an attacker to pass a raw MPLS encoded packet into a network and operators have considerable experience at excluding such packets at the network boundaries.

It is easy for an ingress node to detect any attempt to smuggle IP packet into the network since it would see that the UDP destination port was set to MPLS. SR packets not having a destination address terminating in the network would be transparently carried and would pose no security risk to the network under consideration.

11. IANA Considerations

This document makes no IANA requests.

12. Acknowledgements

This draft was partly inspired by [I-D.xu-mpls-unified-source-routing-instruction], and we acknowledge the following authors of version -02 of that draft: Robert Raszuk, Uma Chunduri, Luis M. Contreras, Luay Jalil, Hamid Assarpour, Gunter Van De Velde, Jeff Tantsura, and Shaowen Ma.

Thanks to Joel Halpern, Bruno Decraene, Loa Andersson, Ron Bonica, Eric Rosen, Robert Raszuk, Wim Henderickx, Jim Guichard, and Gunter Van De Velde for their insightful comments on this draft.

13. Contributors

o Mach Chen, Huawei Technologies, mach.chen@huawei.com

14. References

14.1. Normative References

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-13 (work in progress), October 2017.

[I-D.ietf-spring-segment-routing-mpls]
Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with MPLS data plane", draft-ietf-spring-segment-routing-mpls-10 (work in progress), June 2017.

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

[RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001, <https://www.rfc-editor.org/info/rfc3032>.

[RFC5462] Andersson, L. and R. Asati, "Multiprotocol Label Switching (MPLS) Label Stack Entry: "EXP" Field Renamed to "Traffic Class" Field", RFC 5462, DOI 10.17487/RFC5462, February 2009, <https://www.rfc-editor.org/info/rfc5462>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black, "Encapsulating MPLS in UDP", RFC 7510, DOI 10.17487/RFC7510, April 2015, <https://www.rfc-editor.org/info/rfc7510>.

14.2. Informative References

[I-D.ietf-6man-segment-routing-header]
Previdi, S., Filsfils, C., Raza, K., Leddy, J., Field, B., daniel.voyer@bell.ca, d., daniel.bernier@bell.ca, d., Matsushima, S., Leung, I., Linkova, J., Aries, E., Kosugi, T., Vyncke, E., Lebrun, D., Steinberg, D., and R. Raszuk, "IPv6 Segment Routing Header (SRH)", draft-ietf-6man-segment-routing-header-07 (work in progress), July 2017.

[I-D.ietf-isis-encapsulation-cap]
Xu, X., Decraene, B., Raszuk, R., Chunduri, U., Contreras, L., and L. Jalil, "Advertising Tunnelling Capability in IS-IS", draft-ietf-isis-encapsulation-cap-01 (work in progress), April 2017.

[I-D.ietf-isis-segment-routing-extensions]
Previdi, S., Filsfils, C., Bashandy, A., Gredler, H., Litkowski, S., Decraene, B., and j. jefftant@gmail.com, "IS-IS Extensions for Segment Routing", draft-ietf-isis-segment-routing-extensions-13 (work in progress), June 2017.

[I-D.ietf-ospf-encapsulation-cap]
Xu, X., Decraene, B., Raszuk, R., Contreras, L., and L. Jalil, "The Tunnel Encapsulations OSPF Router Information", draft-ietf-ospf-encapsulation-cap-09 (work in progress), October 2017.

[I-D.ietf-ospf-ospfv3-segment-routing-extensions]
Psenak, P., Previdi, S., Filsfils, C., Gredler, H., Shakir, R., Henderickx, W., and J. Tantsura, "OSPFv3 Extensions for Segment Routing", draft-ietf-ospf-ospfv3-segment-routing-extensions-10 (work in progress), September 2017.

[I-D.ietf-ospf-segment-routing-extensions]
Psenak, P., Previdi, S., Filsfils, C., Gredler, H., Shakir, R., Henderickx, W., and J. Tantsura, "OSPF Extensions for Segment Routing", draft-ietf-ospf-segment-routing-extensions-21 (work in progress), October 2017.

[I-D.ietf-spring-ipv6-use-cases]
Brzozowski, J., Leddy, J., Filsfils, C., Maglione, R., and M. Townsley, "IPv6 SPRING Use Cases", draft-ietf-spring-ipv6-use-cases-11 (work in progress), June 2017.

[I-D.xu-mpls-unified-source-routing-instruction]
Xu, X., Bashandy, A., Assarpour, H., Ma, S., Henderickx, W., and j. jefftant@gmail.com, "Unified Source Routing Instructions using MPLS Label Stack", draft-xu-mpls-unified-source-routing-instruction-04 (work in progress), September 2017.

[RFC2992] Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, DOI 10.17487/RFC2992, November 2000, <https://www.rfc-editor.org/info/rfc2992>.

[RFC3985] Bryant, S., Ed. and P. Pate, Ed., "Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture", RFC 3985, DOI 10.17487/RFC3985, March 2005, <https://www.rfc-editor.org/info/rfc3985>.

[RFC4023] Worster, T., Rekhter, Y., and E. Rosen, Ed., "Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)", RFC 4023, DOI 10.17487/RFC4023, March 2005, <https://www.rfc-editor.org/info/rfc4023>.

Authors' Addresses

Stewart Bryant (editor)
Huawei

Email: stewart.bryant@gmail.com

Adrian Farrel (editor)
Juniper Networks

Email: afarrel@juniper.net

John Drake
Juniper Networks

Email: jdrake@juniper.net

Jeff Tantsura
Individual

Email: jefftant.ietf@gmail.com

MPLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

A. Farrel
Juniper Networks
S. Bryant
Huawei
J. Drake
Juniper Networks
October 30, 2017

An MPLS-Based Forwarding Plane for Service Function Chaining
draft-farrel-mpls-sfc-02

Abstract

Service Function Chaining (SFC) is the process of directing packets through a network so that they can be acted on by an ordered set of abstract service functions before being delivered to the intended destination. An architecture for SFC is defined in RFC7665.

The Network Service Header (NSH) can be inserted into packets to steer them along a specific path to realize a Service Function Chain.

Multiprotocol Label Switching (MPLS) is a widely deployed forwarding technology that uses labels to identify the forwarding actions to be taken at each hop through a network. Segment Routing is a mechanism that provides a source routing paradigm for steering packets in an MPLS network.

This document describes how Service Function Chaining can be achieved in an MPLS network by means of a logical representation of the NSH in an MPLS label stack.

Requirements Language

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].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Service Function Chaining (SFC) is the process of directing packets through a network so that they can be acted on by an ordered set of abstract service functions before being delivered to the intended destination. An architecture for SFC is defined in [RFC7665].

When applying a particular Service Function Chain to the traffic selected by a service classifier, the traffic needs to be steered through an ordered set of Service Functions (SFs) in the network. This ordered set of SFs is termed a Service Function Path (SFP), and the traffic is passed between Service Function Forwarders (SFFs) that are responsible for delivering the packets to the SFs and for forwarding them onward to the next SFF.

In order to steer the selected traffic between SFFs and to the correct SFs the service classifier needs to attach information to each packet. This information indicates the SFP on which the packet is being forwarded and hence the SFs to which it must be delivered. The information also indicates the progress the packet has already made along the SFP.

The Network Service Header (NSH) [I-D.ietf-sfc-nsh] has been defined to carry the necessary information for Service Function Chaining in packets. The NSH can be inserted into packets and contains various information including a Service Path Indicator (SPI), a Service Index (SI), and a Time To Live (TTL) counter.

Multiprotocol Label Switching (MPLS) [RFC3031] is a widely deployed forwarding technology that uses labels to identify the forwarding actions to be taken at each hop through a network. In many cases, MPLS will be used as a tunneling technology to carry packets through networks between SFFs.

Segment Routing [RFC7855] introduces a source routing paradigm into packet switched networks. The application of Segment Routing in MPLS networks is described in [I-D.ietf-spring-segment-routing-mpls] and is known as MPLS-SR.

This document describes how Service Function Chaining can be achieved in an MPLS network by means of a logical representation of the NSH in an MPLS label stack. This approach is applicable to both classical MPLS forwarding (where labels are looked up at each hop, and swapped for the next hop [RFC3031]) and MPLS Segment Routing (where labels are looked up at each hop, and popped to reveal the next label to action [I-D.ietf-spring-segment-routing-mpls]). The mechanisms described in this document are a compromise between the full function that can be achieved using the NSH, and the benefits of reusing the existing MPLS forwarding paradigms.

It is assumed that the reader is fully familiar with the terms and concepts introduced in [RFC7665] and [I-D.ietf-sfc-nsh].

2. Choice of Data Plane SPI/SI Representation

While [I-D.ietf-sfc-nsh] defines the NSH that can be used in a number of environments, this document provides a mechanism to handle situations in which the NSH is not ubiquitously deployed. In this case it is possible to use an alternative data plane representation of the SPI/SI by carrying the identical semantics in MPLS labels.

In order to correctly select the mechanism by which SFC information is encoded and carried between SFFs, it may be necessary to configure the capabilities and choices either within the whole Service Function Overlay Network, or on a hop by hop basis. It is a requirement that both ends of a tunnel over the underlay network know that the tunnel is used for SFC and know what form of NSH representation is used. A control plane signalling approach to achieve these objectives is provided using BGP in [I-D.ietf-bess-nsh-bgp-control-plane].

Note that the encoding of the SFC information is independent of the choice of tunneling technology used between SFFs. Thus, an MPLS representation of the logical NSH (as defined in this document) may be used even if the tunnel between a pair of SFFs is not an MPLS tunnel. Conversely, MPLS tunnels may be used to carry other encodings of the logical NSH (specifically, the NSH itself).

3. Basic Unit of Representation

When an MPLS label stack is used to carry a logical NSH, a basic unit of representation is used. This unit comprises two MPLS labels as shown below. The unit may be present one or more times in the label stack as explained in subsequent sections.

In order to convey the same information as is present in the NSH, two MPLS label stack entries are used. One carries a label to provide context within the SFC scope (the SFC Context Label), and the other carries a label to show which service function is to be actioned (the SF Label). This two-label unit is shown in Figure 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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          SFC Context Label            | TC  |S|       TTL     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          SF Label                     | TC  |S|       TTL     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: The Basic Unit of MPLS Label Stack for SFC

The fields of these two label stack entries are encoded as follows:

Label: The Label fields contain the values of the SFC Context Label and the SF Label encoded as 20 bit integers. The precise semantics of these label fields are dependent on whether the label stack entries are used for MPLS swapping (see Section 4) or MPLS-SR (see Section 5).

TC: The TC bits have no meaning. They SHOULD be set to zero in both label stack entries and MUST be ignored.

S: The bottom of stack flag has its usual meaning in MPLS. It MUST be clear in the SFC Context label stack entry and MAY be set in the SF label stack entry depending on whether the label is the bottom of stack.

TTL: The TTL field in the SFC Context label stack entry SHOULD be set to 1. The TTL in SF label stack entry (called the SF TTL) is set according to its use for MPLS swapping (see Section 4) or MPLS-SR (see Section 5 and is used to mitigate packet loops.

The sections that follow show how this basic unit of MPLS label stack may be used for SFC in the MPLS label swapping case and in the MPLS-SR case. For simplicity, these sections do not describe the use of metadata: that is covered separately in Section 9.

4. MPLS Label Swapping

This section describes how the basic unit of MPLS label stack for SFC introduced in Section 3 is used when MPLS label swapping is in use. As can be seen from Figure 2, the top of the label stack comprises the labels necessary to deliver the packet over the MPLS tunnel between SFFs. Any MPLS encapsulation may be used (i.e., MPLS, MPLS in UDP, MPLS in GRE, and MPLS in VXLAN or GPE), thus the tunnel technology does not need to be MPLS, but that is shown here for simplicity.

An entropy label ([RFC6790]) may also be present as described in Section 8

Under these labels (or other encapsulation) comes a single instance of the basic unit of MPLS label stack for SFC. In addition to the interpretation of the fields of these label stack entries provided in Section 3 the following meanings are applied:

SPI Label: The Label field of the SFC Context label stack entry contains the value of the SPI encoded as a 20 bit integer. The semantics of the SPI is exactly as defined in [I-D.ietf-sfc-nsh]. Note that an SPI as defined by [I-D.ietf-sfc-nsh] can be encoded in 3 octets (i.e., 24 bits), but that the Label field allows for only 20 bits and reserves the values 0 though 15 as 'special purpose' labels [RFC7274]. Thus, a system using MPLS representation of the logical NSH MUST NOT assign SPI values greater than 2^20 - 1 or less than 16.

SI Label: The Label field of the SF label stack entry contains the value of the SI exactly as defined in [I-D.ietf-sfc-nsh]. Since the SI requires only 8 bits, and to avoid overlap with the 'special purpose' label range of 0 through 15 [RFC7274], the SI is carried in the top (most significant) 8 bits of the Label field with the low order 12 bits set to zero.

TC: The TC fields are as described in Section 3.

S: The S fields are as described in Section 3.

TTL: The TTL field in the SPI label stack entry SHOULD be set to 1 as stated in Section 3. The TTL in SF label stack entry is decremented once for each forwarding hop in the SFP, i.e., for each SFF transited, and so mirrors the TTL field in the NSH.

```
       ---------------
      ~ Tunnel Labels ~
      +---------------+
      ~   Optional    ~
      ~ Entropy Label ~
      +---------------+ - - -
      |   SPI Label   |
      +---------------+  Basic unit of MPLS label stack for SFC
      |   SI Label    |
      +---------------+ - - -
      |               |
      ~    Payload    ~
      |               |
       ---------------
```

Figure 2: The MPLS SFC Label Stack

The following processing rules apply to the Label fields:

o When a Classifier inserts a packet onto an SFP it sets the SPI Label to indicate the identity of the SFP, and sets the SI Label to indicate the first SF in the path.

o When a component of the SFC system processes a packet it uses the SPI Label to identify the SFP and the SI Label to determine to which SFF or SFI to deliver the packet. Under normal circumstances (with the exception of branching and reclassification - see [I-D.ietf-bess-nsh-bgp-control-plane]) the SPI Label value is preserved on all packets. The SI Label value is modified by SFFs and through reclassification to indicate the next hop along the SFP.

The following processing rules apply to the TTL field of the SF label stack entry, and are derived from section 2.2 of [I-D.ietf-sfc-nsh]:

o When a Classifier places a packet onto an SFP it MUST set the TTL to a value between 1 and 255. It SHOULD set this according to the expected length of the SFP (i.e., the number of SFs on the SFP), but it MAY set it to a larger value according to local configuration. The maximum TTL value supported in an NSH is 63, and so the practical limit here may also be 63.

o When an SFF receives a packet from any component of the SFC system (Classifier, SFI, or another SFF) it MUST discard any packets with TTL set to zero. It SHOULD log such occurrences, but MUST apply rate limiting to any such logs.

- An SFF MUST decrement the TTL by one each time it performs a forwarding lookup.

- If an SFF decrements the TTL to zero it MUST NOT send the packet, and MUST discard the packet. It SHOULD log such occurrences, but MUST apply rate limiting to any such logs.

- SFIs MUST ignore the TTL, but MUST mirror it back to the SFF unmodified along with the SI (which may have been changed by local reclassification).

- If a Classifier along the SFP makes any change to the intended path of the packet including for looping, jumping, or branching (see [I-D.ietf-bess-nsh-bgp-control-plane] it MUST NOT change the SI TTL of the packet. In particular, each component of the SFC system MUST NOT increase the SI TTL value otherwise loops may go undetected.

5. MPLS Segment Routing

This section describes how the basic unit of MPLS label stack for SFC introduced in Section 3 is used when in an MPLS-SR network. As can be seen Figure 3, the top of the label stack comprises the labels necessary to deliver the packet over the MPLS tunnel between SFFs. Any MPLS encapsulation may be used and the tunnel technology does not need to be MPLS or MPLS-SR, but MPLS-SR is shown here for simplicity.

An entropy label ([RFC6790]) may also be present as described in Section 8

Under these labels (or other encapsulation) comes one of more instances of the basic unit of MPLS label stack for SFC. In addition to the interpretation of the fields of these label stack entries provided in Section 3 the following meanings are applied:

SFC Context Label: The Label field of the SFC Context label stack entry contains a label that delivers SFC context. This label may be used to indicate the SPI encoded as a 20 bit integer using the semantics of the SPI is exactly as defined in [I-D.ietf-sfc-nsh] and noting that in this case a system using MPLS representation of the logical NSH MUST NOT assign SPI values greater than 2^20 - 1 or less than 16. This label may also be used to convey other SFC context-speific semantics such as indicating, perhaps with a node SID (see [I-D.ietf-spring-segment-routing]), how to interpret the SF Label.

SF Label: The Label field of the SF label stack entry contains a value that identifies the next SFI to be actioned for the packet.

This label may be scoped globally or within the context of the preceding SFC Context Label and comes from the range 16 ... 2^20 - 1.

TC: The TC fields are as described in Section 3.

S: The S fields are as described in Section 3.

TTL: The TTL field in the SFC Context label stack entry SHOULD be set to 1 as stated in Section 3. The TTL in SF label stack entry is set according to the norms for MPLS-SR.

```
       -------------------
      ~   MPLS-SR Labels  ~
      +-------------------+
      ~     Optional      ~
      ~   Entropy Label   ~
      +-------------------+ - - -
      | SFC Context Label |
      +-------------------+  Basic unit of MPLS label stack for SFC
      |     SF Label      |
      +-------------------+ - - -
      | SFC Context Label |
      +-------------------+  Basic unit of MPLS label stack for SFC
      |     SF Label      |
      +-------------------+ - - -
      ~                   ~
      +-------------------+ - - -
      | SFC Context Label |
      +-------------------+  Basic unit of MPLS label stack for SFC
      |     SF Label      |
      +-------------------+ - - -
      |                   |
      ~    Payload        ~
      |                   |
       -------------------
```

Figure 3: The MPLS SFC Label Stack for Segment Routing

The following processing rules apply to the Label fields:

o When a Classifier inserts a packet onto an SFP it adds a stack comprising one or more instances of the basic unit of MPLS label stack for SFC. Taken together, this stack defines the SFs to be actioned and so defines the SFP that the packet will traverse.

- When a component of the SFC system processes a packet it uses the top basic unit of label stack for SFC to determine to which SFI to next deliver the packet. When an SFF receives a packet it examines the top basic unit of MPLS label stack for SFC to determine where to send the packet next. If the next recipient is a local SFI, the SFC strips the basic unit of MPLS label stack for SFC before forwarding the packet.

6. Mixed Mode Forwarding

The previous sections describe homogeneous networks where SFC forwarding is either all label swapping or all label popping. But it is also possible that different parts of the network utilize swapping or popping for different purposes.

When an SFF receives a packet containing an MPLS label stack, it checks whether it is processing an {SFP, SI} label pair for label swapping or a {context label, SFI index} label pair for MPLS-SR. It then selects the appropriate SFI to which to send the packet. When it receives the packet back from the SFI, it has four cases to consider.

- If the current hop requires an {SFP, SI} and the next hop requires an {SFP, SI}, it sets the SI label to the SI value of the current hop, selects an instance of the SF to be executed at the next hop, and tunnels the packet to the SFF for that SFI.

- If the current hop requires an {SFP, SI} and the next hop requires a {context label, SFI label}, it pops the {SFP, SI} from the top of the MPLS label stack and tunnels the packet to the SFF indicated by the context label.

- If the current hop requires a {context label, SFI label}, it pops the {context label, SFI label} from the top of the MPLS label stack.

    * If the new top of the MPLS label stack contains an {SFP, SI} label pair, it selects an SFI to use at the next hop, and tunnels the packet to SFF for that SFI.

    * If the top of the MPLS label stack contains a {context label, SFI label}, it tunnels the packet to the SFF indicated by the context label.

7. Control Plane Considerations

In order that a packet may be forwarded along an SFP several functional elements must be executed.

- o Discovery/advertisement of SFIs.
- o Computation of SFP.
- o Programming of Classifiers.
- o Advertisement of forwarding instructions.

Various approaches may be taken. These include a fully centralized model where SFFs report to a central controller the SFIs that they support, the central controller computes the SFP and programs the Classifiers, and (if the label swapping approach is taken) the central controller installs forwarding state in the SFFs that lie on the SFP.

Alternatively, a dynamic control plane may be used such as that described in [I-D.ietf-bess-nsh-bgp-control-plane]. In this case the SFFs use the control plane to advertise the SFIs that they support, a central controller computes the SFP and programs the Classifiers, and (if the label swapping approach is taken) the central controller uses the control plane to advertise the SFPs so that SFFs that lie on the SFP can install the necessary forwarding state.

8. Use of the Entropy Label

Entropy is used in ECMP situations to ensure that packets from the same flow travel down the same path, thus avoiding jitter or re-ordering issues within a flow.

Entropy is often determined by hashing on specific fields in a packet header such as the "five-tuple" in the IP and transport headers. However, when an MPLS label stack is present, the depth of the stack could be too large for some processors to correctly determine the entropy hash. This problem is addressed by the inclusion of an Entropy Label as described in [RFC6790].

When entropy is desired for packets as they are carried in MPLS tunnels over the underlay network, it is RECOMMENDED that an Entropy Label is included in the label stack immediately after the tunnel labels and before the SFC labels as shown in Figure 2 and Figure 3.

If an Entropy Label is present in a packet received by an SR-capabale node (at the end of a tunnel across the underlay network), it is

RECOMMENDED that the value of that label is preserved and used in an Entropy Label inserted in the label stack when the packet is forwarded (on the next tunnel) to the next SFF.

If an Entropy Label is present in an MPLS payload, it is RECOMMENDED that the initial Classifier use that value in an Entropy Label inserted in the label stack when the packet is forwarded (on the first tunnel) to the first SFF. In this case it is not necessary to remove the Entropy Label from the payload.

9. Metadata

Metadata is defined in [RFC7665] as providing "the ability to exchange context information between classifiers and SFs, and among SFs." [I-D.ietf-sfc-nsh] defines how this context information can be directly encoded in fields that form part of the NSH encapsulation.

The next two sections describe how metadata is associated with user data packets, and how metadata may is exchanged between SFC nodes in the network, when using an MPLS encoding of the logical representation of the NSH.

9.1. Indicating Metadata in User Data Packets

Metadata is achieved in the MPLS realization of the logical NSH by the use of an SFC Metadata Label which uses the Extended Special Purpose Label construct [RFC7274]. Thus, three label stack entries are present as shown in Figure 4:

o The Extension Label (value 15)

o An extended special purpose label called the Metadata Label Indicator (MLI) (value TBD1 by IANA)

o The Metadata Label (ML).

```
 ----------------
| Extension = 15 |
+----------------+
|      MLI       |
+----------------+
| Metadata Label |
 ---------------
```

Figure 4: The MPLS SFC Metadata Label

The Metadata Label value is an index into a table of metadata that is programmed into the network using in-band or out-of-band mechanisms. Out-of-band mechanisms potentially include management plane and control plane solutions (such as [I-D.ietf-bess-nsh-bgp-control-plane]), but are out of scope for this document. The in-band mechanism is described in Section 9.2

The SFC Metadata Label (as a set of three labels as indicated in Figure 4) may be present zero, one, or more times in an MPLS SFC packet. For MPLS label swapping, the SFC Metadata Labels are placed immediately after the basic unit of MPLS label stack for SFC as shown in Figure 5. For MPLS-SR, the SFC Metadata Labels can be present zero, one, or more times and are placed at the bottom of the label stack as shown in Figure 6.

```
       ----------------
      ~ Tunnel Labels  ~
      +----------------+
      ~   Optional     ~
      ~ Entropy Label  ~
      +----------------+
      |   SPI Label    |
      +----------------+
      |   SI Label     |
      +----------------+
      | Extension = 15 |
      +----------------+
      |      MLI       |
      +----------------+
      | Metadata Label |
      +----------------+
      ~     Other      ~
      |    Metadata    |
      ~     Labels     ~
      +----------------+
      |                |
      ~    Payload     ~
      |                |
       ----------------
```

Figure 5: The MPLS SFC Label Stack for Label Swapping with Metadata Label

```
       -------------------
      ~   MPLS-SR Labels  ~
      +--------------------+
      ~     Optional       ~
      ~   Entropy Label    ~
      +--------------------+
      | SFC Context Label |
      +--------------------+
      |      SF Label      |
      +--------------------+
      ~                    ~
      +--------------------+
      | SFC Context Label |
      +--------------------+
      |      SF Label      |
      +--------------------+
      |   Extension = 15   |
      +--------------------+
      |        MLI         |
      +--------------------+
      |  Metadata Label    |
      +--------------------+
      ~       Other        ~
      |      Metadata      |
      ~       Labels       ~
      +--------------------+
      |                    |
      ~    Payload         ~
      |                    |
       -------------------
```

Figure 6: The MPLS SFC Label Stack for MPLS-SR with Metadata Label

9.2. Inband Programming of Metadata

A mechanism for sending metadata associated with an SFP without a payload packet is described in [I-D.farrel-sfc-convent]. The same approach can be used in an MPLS network where the NSH is logically represented by an MPLS label stack.

The packet header is formed exactly as previously described in this document so that the packet will follow the SFP through the SFC network. However, instead of payload data, metadata is included after the bottom of the MPLS label stack. An Extended Special Purpose Label is used to indicate that the metadata is present. Thus, three label stack entries are present:

- o The Extension Label (value 15)
- o An extended special purpose label called the Metadata Present Indicator (MPI) (value TBD2 by IANA)
- o The Metadata Label (ML) that is associated with this metadata on this SFP and can be used to indicate the use of the metadata as described in Section 9.

The SFC Metadata Present Label, if present, is placed immediately after the last basic unit of MPLS label stack for SFC. The resultant label stacks are shown in Figure 7 for the MPLS label swapping case and Figure 8 for the MPLS-SR case.

```
         ---------------
        ~ Tunnel Labels ~
        +---------------+
        ~   Optional    ~
        ~ Entropy Label ~
        +---------------+
        |   SPI Label   |
        +---------------+
        |   SI Label    |
        +---------------+
        | Extension = 15|
        +---------------+
        |      MPI      |
        +---------------+
        | Metadata Label|
        +---------------+
        |               |
        ~    Metadata   ~
        |               |
         ---------------
```

Figure 7: The MPLS SFC Label Stack Carrying Metadata

```
      -------------------
     ~   MPLS-SR Labels  ~
     +-------------------+
     ~     Optional      ~
     ~   Entropy Label   ~
     +-------------------+
     | SFC Context Label |
     +-------------------+
     |     SF Label      |
     +-------------------+
     | SFC Context Label |
     +-------------------+
     |     SF Label      |
     +-------------------+
     ~                   ~
     +-------------------+
     | SFC Context Label |
     +-------------------+
     |     SF Label      |
     +-------------------+
     |   Extension = 15  |
     +-------------------+
     |        MPI        |
     +-------------------+
     |  Metadata Label   |
     +-------------------+
     |                   |
     ~    Metadata       ~
     |                   |
      -------------------
```

Figure 8: The MPLS SFC Label Stack for MPLS-SR Carrying Metadata

In both cases the metadata is formatted as a TLV as shown in Figure 9.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            Length             |         Metadata Type         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                            Metadata                           ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9: The Metadata TLV

The fields of this TLV are interpreted as follows:

Length: The length of the metadata carried in the Metadata field in octets not including any padding.

Metadata Type: The type of the metadata present. Values for this field are taken from the "MD Types" registry maintained by IANA and defined in [I-D.ietf-sfc-nsh].

Metadata: The actual metadata formatted as described in whatever document defines the metadata. This field is end-padded with zero to three octets of zeroes to take it up to a four octet boundary.

10. Worked Examples

Consider the simplistic MPLS SFC overlay network shown in Figure 10. A packet is classified for an SFP that will see it pass through two Service Functions, SFa and SFb, that are accessed through Service Function Forwarders SFFa and SFFb respectively. The packet is ultimately delivered to destination, D.

Let us assume that the SFP is computed and assigned the SPI of 239. The forwarding details of the SFP are distributed (perhaps using the mechanisms of [I-D.ietf-bess-nsh-bgp-control-plane]) so that the SFFs are programmed with the necessary forwarding instructions.

The packet progresses as follows:

a. The Classifier assigns the packet to the SFP and imposes two label stack entries comprising a single basic unit of MPLS SFC representation:

   * The higher label stack entry contains a label carrying the SPI value of 239.

   * The lower label stack entry contains a label carrying the SI value of 255.

   Further labels may be imposed to tunnel the packet from the Classifier to SFFa.

b. When the packet arrives at SFFa it strips any labels associated with the tunnel that runs from the Classifier to SFFa. SFFa examines the top labels and matches the SPI/SI to identify that the packet should be forwarded to SFa. The packet is forwarded to SFa unmodified.

c. SFa performs its designated function and returns the packet to SFFa.

d. SFFa modifies the SI in the lower label stack entry (to 254) and uses the SPI/SI to look up the forwarding instructions. It sends the packet with two label stack entries:

   * The higher label stack entry contains a label carrying the SPI value of 239.

   * The lower label stack entry contains a label carrying the SI value of 254.

   Further labels may be imposed to tunnel the packet from the SFFa to SFFb.

e. When the packet arrives at SFFb it strips any labels associated with the tunnel from SFFa. SFFb examines the top labels and matches the SPI/SI to identify that the packet should be forwarded to SFb. The packet is forwarded to SFb unmodified.

f. SFb performs its designated function and returns the packet to SFFb.

g. SFFb modifies the SI in the lower label stack entry (to 253) and uses the SPI/SI to lookup up the forwarding instructions. It determines that it is the last SFF in the SFP so it strips the two SFC label stack entries and forwards the payload toward D using the payload protocol.

```
        +---------------------------------------------------+
        |                   MPLS SFC Network                |
        |                                                   |
        |            +---------+       +---------+          |
        |            |   SFa   |       |   SFb   |          |
        |            +----+----+       +----+----+          |
        |               ^ | |             ^ | |             |
        |            (b)| | |(c)       (e)| | |(f)          |
        |       (a)     | | V     (d)     | | V    (g)      |
   +----------+ ---> +----+----+ ----> +----+----+ ---> +--------+
   |Classifier+------+  SFFa   +-------+  SFFb   +------+   D    |
   +----------+      +---------+       +---------+      +--------+
        |                                                   |
        +---------------------------------------------------+
```

Figure 10: Service Function Chaining in an MPLS Network

Alternatively, consider the MPLS SFC overlay network shown in Figure 11. A packet is classified for an SFP that will see it pass through two Service Functions, SF1 and SF2, that are accessed through Service Function Forwarders SFF1 and SFF2 respectively. The packet is ultimately delivered to destination, D.

Let us assume that the SFP is computed and assigned the SPI of 239. However, the forwarding state for the SFP is not distributed and installed in the network. Instead it will be attached to the individual packets using MPLS-SR.

The packet progresses as follows:

1. The Classifier assigns the packet to the SFP and imposes two basic units of MPLS SFC representation to describe the full SFP:

   * The top basic unit comprises two label stack entries as follows:

     + The higher label stack entry contains a label carrying the SFC context.

     + The lower label stack entry contains a label carrying the SF indicator for SF1.

   * The lower basic unit comprises two label stack entries as follows:

     + The higher label stack entry contains a label carrying the SFC context.

     + The lower label stack entry contains a label carrying the SF indicator for SF2.

   Further labels may be imposed to tunnel the packet from the Classifier to SFF1.

2. When the packet arrives at SFF1 it strips any labels associated with the tunnel from the Classifier. SFF1 examines the top labels and matches the context/SF values to identify that the packet should be forwarded to SF1. The packet is forwarded to SF1 unmodified.

3. SF1 performs its designated function and returns the packet to SFF1.

4. SFF1 strips the top basic unit of MPLS SFC representation revealing the next basic unit. It then uses the revealed

context/SF values to determine how to route the packet to the next SFF, SFF2. It sends the packet with just one basic unit of MPLS SFC representation comprising two label stack entries:

* The higher label stack entry contains a label carrying the SFC context.

* The lower label stack entry contains a label carrying the SF indicator for SF2.

Further labels may be imposed to tunnel the packet from the SFF1 to SFF2.

5. When the packet arrives at SFF2 it strips any labels associated with the tunnel from SFF1. SFF2 examines the top labels and matches the context/SF values to identify that the packet should be forwarded to SF2. The packet is forwarded to SF2 unmodified.

6. SF2 performs its designated function and returns the packet to SFF2.

7. SFF2 strips the top basic unit of MPLS SFC representation revealing the payload packet. It forwards the payload toward D using the payload protocol.

```
        +---------------------------------------------------+
        |                 MPLS-SR SFC Network               |
        |                                                   |
        |            +---------+       +---------+          |
        |            |   SF1   |       |   SF2   |          |
        |            +----+----+       +----+----+          |
        |               ^ | |             ^ | |             |
        |            (2)| | |(3)       (5)| | |(6)          |
        |       (1)     | | V     (4)     | | V    (7)      |
   +----------+ ---> +----+----+ ----> +----+----+ ---> +-------+
   |Classifier+------+  SFF1   +-------+  SFF2   +------+   D   |
   +----------+      +---------+       +---------+      +-------+
        |                                                   |
        +---------------------------------------------------+
```

Figure 11: Service Function Chaining in an MPLS-SR Network

11. Security Considerations

Discussion of the security properties of SFC networks can be found in [RFC7665]. Further security discussion for the NSH and its use is present in [I-D.ietf-sfc-nsh].

It is fundamental to the SFC design that the classifier is a trusted resource which determines the processing that the packet will be subject to, including for example the firewall. It is also fundamental to the Segment Routing design that packets are routed through the network using the path specified by the node imposing the SIDs. Where an SF is not encapsulation aware the packet may exist as an IP packet, however this is an intrinsic part of the SFC design which needs to define how a packet is protected in that environment. Where a tunnel is used to link two non-MPLS domains, the tunnel design needs to specify how it is secured. Thus the security vulnerabilities are addressed in the underlying technologies used by this design, which itself does not introduce any new security vulnerabilities.

12. IANA Considerations

This document requests IANA to make allocations from the "Extended Special-Purpose MPLS Label Values" subregistry of the "Special-Purpose Multiprotocol Label Switching (MPLS) Label Values" registry as follows:

Value	Description	
TBD1	Metadata Label Indicator (MLI)	[This.I-D]
TBD2	Metadata Present Indicator (MPI)	[This.I-D]

13. Acknowledgements

This document derives ideas and text from [I-D.ietf-bess-nsh-bgp-control-plane].

The authors are grateful to all those who contributed to the discussions that led to this work: Loa Andersson, Andrew G. Malis, Alexander Vainshtein, Joel M. Halpern, Tony Przygienda, Stuart Mackie, Keyur Patel, and Jim Guichard.

14. References

14.1. Normative References

[I-D.ietf-sfc-nsh]
Quinn, P., Elzur, U., and C. Pignataro, "Network Service Header (NSH)", draft-ietf-sfc-nsh-27 (work in progress), October 2017.

[I-D.ietf-spring-segment-routing-mpls]
Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with MPLS data plane", draft-ietf-spring-segment-routing-mpls-10 (work in progress), June 2017.

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

[RFC7274] Kompella, K., Andersson, L., and A. Farrel, "Allocating and Retiring Special-Purpose MPLS Labels", RFC 7274, DOI 10.17487/RFC7274, June 2014, <https://www.rfc-editor.org/info/rfc7274>.

14.2. Informative References

[I-D.farrel-sfc-convent]
Farrel, A. and J. Drake, "Operating the Network Service Header (NSH) with Next Protocol "None"", draft-farrel-sfc-convent-03 (work in progress), October 2017.

[I-D.ietf-bess-nsh-bgp-control-plane]
Farrel, A., Drake, J., Rosen, E., Uttaro, J., and L. Jalil, "BGP Control Plane for NSH SFC", draft-ietf-bess-nsh-bgp-control-plane-01 (work in progress), September 2017.

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-13 (work in progress), October 2017.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <https://www.rfc-editor.org/info/rfc3031>.

[RFC6790] Kompella, K., Drake, J., Amante, S., Henderickx, W., and L. Yong, "The Use of Entropy Labels in MPLS Forwarding", RFC 6790, DOI 10.17487/RFC6790, November 2012, <https://www.rfc-editor.org/info/rfc6790>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

[RFC7855] Previdi, S., Ed., Filsfils, C., Ed., Decraene, B., Litkowski, S., Horneffer, M., and R. Shakir, "Source Packet Routing in Networking (SPRING) Problem Statement and Requirements", RFC 7855, DOI 10.17487/RFC7855, May 2016, <https://www.rfc-editor.org/info/rfc7855>.

Authors' Addresses

Adrian Farrel
Juniper Networks

Email: afarrel@juniper.net

Stewart Bryant
Huawei

Email: stewart.bryant@gmail.com

John Drake
Juniper Networks

Email: jdrake@juniper.net

MPLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 23, 2018

A. Farrel
Juniper Networks
S. Bryant
Huawei
J. Drake
Juniper Networks

March 22, 2018

An MPLS-Based Forwarding Plane for Service Function Chaining
draft-farrel-mpls-sfc-05

Abstract

Service Function Chaining (SFC) is the process of directing packets through a network so that they can be acted on by an ordered set of abstract service functions before being delivered to the intended destination. An architecture for SFC is defined in RFC7665.

The Network Service Header (NSH) can be inserted into packets to steer them along a specific path to realize a Service Function Chain.

Multiprotocol Label Switching (MPLS) is a widely deployed forwarding technology that uses labels placed in a packet in a label stack to identify the forwarding actions to be taken at each hop through a network. Actions may include swapping or popping the labels as well, as using the labels to determine the next hop for forwarding the packet. Labels may also be used to establish the context under which the packet is forwarded.

This document describes how Service Function Chaining can be achieved in an MPLS network by means of a logical representation of the NSH in an MPLS label stack. It does not deprecate or replace the NSH, but acknowledges that there may be a need for an interim deployment of SFC functionality in brownfield networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Service Function Chaining (SFC) is the process of directing packets through a network so that they can be acted on by an ordered set of abstract service functions before being delivered to the intended destination. An architecture for SFC is defined in [RFC7665].

When applying a particular Service Function Chain to the traffic selected by a service classifier, the traffic needs to be steered through an ordered set of Service Functions (SFs) in the network. This ordered set of SFs is termed a Service Function Path (SFP), and the traffic is passed between Service Function Forwarders (SFFs) that are responsible for delivering the packets to the SFs and for forwarding them onward to the next SFF.

In order to steer the selected traffic between SFFs and to the correct SFs the service classifier needs to attach information to each packet. This information indicates the SFP on which the packet is being forwarded and hence the SFs to which it must be delivered. The information also indicates the progress the packet has already made along the SFP.

The Network Service Header (NSH) [RFC8300] has been defined to carry the necessary information for Service Function Chaining in packets. The NSH can be inserted into packets and contains various information including a Service Path Indicator (SPI), a Service Index (SI), and a Time To Live (TTL) counter.

Multiprotocol Label Switching (MPLS) [RFC3031] is a widely deployed forwarding technology that uses labels placed in a packet in a label stack to identify the forwarding actions to be taken at each hop through a network. Actions may include swapping or popping the labels as well, as using the labels to determine the next hop for forwarding the packet. Labels may also be used to establish the context under which the packet is forwarded. In many cases, MPLS will be used as a tunneling technology to carry packets through networks between SFFs.

This document describes how Service Function Chaining can be achieved in an MPLS network by means of a logical representation of the NSH in an MPLS label stack. This approach is applicable to all forms of MPLS forwarding (where labels are looked up at each hop, and swapped or popped [RFC3031]). It does not deprecate or replace the NSH, but acknowledges that there may be a need for an interim deployment of SFC functionality in brownfield networks. The mechanisms described in this document are a compromise between the full function that can be achieved using the NSH, and the benefits of reusing the existing MPLS forwarding paradigms.

It is assumed that the reader is fully familiar with the terms and concepts introduced in [RFC7665] and [RFC8300].

Note that one of the features of the SFC architecture described in [RFC7665] is the "SFC proxy" that exists to include legacy SFs that are not able to process NSH-encapsulated packets. This issue is equally applicable to the use of MPLS-encapsulated packets that encode a logical representation of an NSH. It is discussed further in Section 8.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Choice of Data Plane SPI/SI Representation

While [RFC8300] defines the NSH that can be used in a number of environments, this document provides a mechanism to handle situations in which the NSH is not ubiquitously deployed. In this case it is possible to use an alternative data plane representation of the SPI/ SI by carrying the identical semantics in MPLS labels.

In order to correctly select the mechanism by which SFC information is encoded and carried between SFFs, it may be necessary to configure the capabilities and choices either within the whole Service Function Overlay Network, or on a hop by hop basis. It is a requirement that both ends of a tunnel over the underlay network (i.e., a pair of SFFs adjacent in the SFC) know that the tunnel is used for SFC and know what form of NSH representation is used. A control plane signalling approach to achieve these objectives is provided using BGP in [I-D.ietf-bess-nsh-bgp-control-plane].

Note that the encoding of the SFC information is independent of the choice of tunneling technology used between SFFs. Thus, an MPLS representation of the logical NSH (as defined in this document) may be used even if the tunnel between a pair of SFFs is not an MPLS tunnel. Conversely, MPLS tunnels may be used to carry other encodings of the logical NSH (specifically, the NSH itself).

4. Basic Unit of Representation

When an MPLS label stack is used to carry a logical NSH, a basic unit of representation is used. This unit comprises two MPLS labels as shown below. The unit may be present one or more times in the label stack as explained in subsequent sections.

In order to convey the same information as is present in the NSH, two MPLS label stack entries are used. One carries a label to provide context within the SFC scope (the SFC Context Label), and the other carries a label to show which service function is to be actioned (the SF Label). This two-label unit is shown in Figure 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            SFC Context Label          | TC  |S|       TTL     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            SF Label                   | TC  |S|       TTL     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: The Basic Unit of MPLS Label Stack for SFC

The fields of these two label stack entries are encoded as follows:

Label: The Label fields contain the values of the SFC Context Label and the SF Label encoded as 20 bit integers. The precise semantics of these label fields are dependent on whether the label stack entries are used for MPLS label swapping (see Section 5) or MPLS label stacking (see Section 6).

TC: The TC bits have no meaning. They SHOULD be set to zero in both label stack entries when a packet is sent and MUST be ignored on receipt.

S: The bottom of stack bit has its usual meaning in MPLS. It MUST be clear in the SFC Context label stack entry and MAY be set in the SF label stack entry depending on whether the label is the bottom of stack.

TTL: The TTL field in the SFC Context label stack entry SHOULD be set to 1. The TTL in SF label stack entry (called the SF TTL) is set according to its use for MPLS label swapping (see Section 5) or MPLS label stacking (see Section 6 and is used to mitigate packet loops.

The sections that follow show how this basic unit of MPLS label stack may be used for SFC in the MPLS label swapping case and in the MPLS label stacking. For simplicity, these sections do not describe the use of metadata: that is covered separately in Section 11.

5. MPLS Label Swapping

This section describes how the basic unit of MPLS label stack for SFC introduced in Section 4 is used when MPLS label swapping is in use. As can be seen from Figure 2, the top of the label stack comprises the labels necessary to deliver the packet over the MPLS tunnel between SFFs. Any MPLS encapsulation may be used (i.e., MPLS, MPLS in UDP, MPLS in GRE, and MPLS in VXLAN or GPE), thus the tunnel technology does not need to be MPLS, but that is shown here for simplicity.

An entropy label ([RFC6790]) may also be present as described in Section 10

Under these labels (or other encapsulation) comes a single instance of the basic unit of MPLS label stack for SFC. In addition to the interpretation of the fields of these label stack entries provided in Section 4 the following meanings are applied:

SPI Label: The Label field of the SFC Context label stack entry contains the value of the SPI encoded as a 20 bit integer. The semantics of the SPI is exactly as defined in [RFC8300]. Note that an SPI as defined by [RFC8300] can be encoded in 3 octets (i.e., 24 bits), but that the Label field allows for only 20 bits and reserves the values 0 though 15 as 'special purpose' labels [RFC7274]. Thus, a system using MPLS representation of the logical NSH MUST NOT assign SPI values greater than 2^20 - 1 or less than 16.

SI Label: The Label field of the SF label stack entry contains the value of the SI exactly as defined in [RFC8300]. Since the SI requires only 8 bits, and to avoid overlap with the 'special purpose' label range of 0 through 15 [RFC7274], the SI is carried in the top (most significant) 8 bits of the Label field with the low order 12 bits set to zero.

TC: The TC fields are as described in Section 4.

S: The S bits are as described in Section 4.

TTL: The TTL field in the SPI label stack entry SHOULD be set to 1 as stated in Section 4. The TTL in SF label stack entry is decremented once for each forwarding hop in the SFP, i.e., for each SFF transited, and so mirrors the TTL field in the NSH.

```
      ---------------
     ~ Tunnel Labels ~
     +---------------+
     ~   Optional    ~
     ~ Entropy Label ~
     +---------------+ - - -
     |   SPI Label   |
     +---------------+  Basic unit of MPLS label stack for SFC
     |   SI Label    |
     +---------------+ - - -
     |               |
     ~    Payload    ~
     |               |
      ---------------
```

Figure 2: The MPLS SFC Label Stack

The following processing rules apply to the Label fields:

o When a Classifier inserts a packet onto an SFP it sets the SPI Label to indicate the identity of the SFP, and sets the SI Label to indicate the first SF in the path.

o When a component of the SFC system processes a packet it uses the SPI Label to identify the SFP and the SI Label to determine to which SFF or instance of an SF (an SFI) to deliver the packet. Under normal circumstances (with the exception of branching and reclassification - see [I-D.ietf-bess-nsh-bgp-control-plane]) the SPI Label value is preserved on all packets. The SI Label value is modified by SFFs and through reclassification to indicate the next hop along the SFP.

The following processing rules apply to the TTL field of the SF label stack entry, and are derived from section 2.2 of [RFC8300]:

o When a Classifier places a packet onto an SFP it MUST set the TTL to a value between 1 and 255. It SHOULD set this according to the expected length of the SFP (i.e., the number of SFs on the SFP), but it MAY set it to a larger value according to local configuration. The maximum TTL value supported in an NSH is 63, and so the practical limit here may also be 63.

o When an SFF receives a packet from any component of the SFC system (Classifier, SFI, or another SFF) it MUST discard any packets with TTL set to zero. It SHOULD log such occurrences, but MUST apply rate limiting to any such logs.

- o An SFF MUST decrement the TTL by one each time it performs a forwarding lookup.

- o If an SFF decrements the TTL to zero it MUST NOT send the packet, and MUST discard the packet. It SHOULD log such occurrences, but MUST apply rate limiting to any such logs.

- o SFIs MUST ignore the TTL, but MUST mirror it back to the SFF unmodified along with the SI (which may have been changed by local reclassification).

- o If a Classifier along the SFP makes any change to the intended path of the packet including for looping, jumping, or branching (see [I-D.ietf-bess-nsh-bgp-control-plane] it MUST NOT change the SI TTL of the packet. In particular, each component of the SFC system MUST NOT increase the SI TTL value otherwise loops may go undetected.

6. MPLS Label Stacking

This section describes how the basic unit of MPLS label stack for SFC introduced in Section 4 is used when MPLS label stacking is used to carry information about the SFP and SFs to be executed. As can be seen in Figure 3, the top of the label stack comprises the labels necessary to deliver the packet over the MPLS tunnel between SFFs. Any MPLS encapsulation may be used.

An entropy label ([RFC6790]) may also be present as described in Section 10

Under these labels comes one of more instances of the basic unit of MPLS label stack for SFC. In addition to the interpretation of the fields of these label stack entries provided in Section 4 the following meanings are applied:

SFC Context Label: The Label field of the SFC Context label stack entry contains a label that delivers SFC context. This label may be used to indicate the SPI encoded as a 20 bit integer using the semantics of the SPI is exactly as defined in [RFC8300] and noting that in this case a system using MPLS representation of the logical NSH MUST NOT assign SPI values greater than $2^20 - 1$ or less than 16. This label may also be used to convey other SFC context-speific semantics such as indicating how to interpret the SF Label or how to forward the packet to the node that offers the SF.

SF Label: The Label field of the SF label stack entry contains a value that identifies the next SFI to be actioned for the packet.

      This label may be scoped globally or within the context of the
      preceding SFC Context Label and comes from the range 16 ... 2^20 -
      1.

   TC: The TC fields are as described in Section 4.

   S: The S bits are as described in Section 4.

   TTL: The TTL fields in the SFC Context label stack entry SF label
      stack entry SHOULD be set to 1 as stated in Section 4, but MAY be
      set to larger values if the label indicated a forwarding operation
      towards the node that hosts the SF.

```
     -------------------
    ~   Tunnel Labels   ~
    +-------------------+
    ~     Optional      ~
    ~   Entropy Label   ~
    +-------------------+ - - -
    | SFC Context Label |
    +-------------------+  Basic unit of MPLS label stack for SFC
    |     SF Label      |
    +-------------------+ - - -
    | SFC Context Label |
    +-------------------+  Basic unit of MPLS label stack for SFC
    |     SF Label      |
    +-------------------+ - - -
    ~                   ~
    +-------------------+ - - -
    | SFC Context Label |
    +-------------------+  Basic unit of MPLS label stack for SFC
    |     SF Label      |
    +-------------------+ - - -
    |                   |
    ~    Payload        ~
    |                   |
     -------------------
```

Figure 3: The MPLS SFC Label Stack for Label Stacking

   The following processing rules apply to the Label fields:

   o When a Classifier inserts a packet onto an SFP it adds a stack
      comprising one or more instances of the basic unit of MPLS label
      stack for SFC. Taken together, this stack defines the SFs to be
      actioned and so defines the SFP that the packet will traverse.

- When a component of the SFC system processes a packet it uses the top basic unit of label stack for SFC to determine to which SFI to next deliver the packet. When an SFF receives a packet it examines the top basic unit of MPLS label stack for SFC to determine where to send the packet next. If the next recipient is a local SFI, the SFC strips the basic unit of MPLS label stack for SFC before forwarding the packet.

7. Mixed Mode Forwarding

The previous sections describe homogeneous networks where SFC forwarding is either all label swapping or all label popping (stacking). But it is also possible that different parts of the network utilize swapping or popping. It is also worth noting that a Classifier may be content to use an SFP as installed in the network by a control plane or management plane and so would use label swapping, but that there may be a point in the SFP where a choice of SFIs can be made (perhaps for load balancing) and where, in this instance, the Classifier wishes to exert control over that choice by use of a specific entry on the label stack.

When an SFF receives a packet containing an MPLS label stack, it checks whether it is processing an {SFP, SI} label pair for label swapping or a {context label, SFI index} label pair for label stacking. It then selects the appropriate SFI to which to send the packet. When it receives the packet back from the SFI, it has four cases to consider.

- If the current hop requires an {SFP, SI} and the next hop requires an {SFP, SI}, it sets the SI label to the SI value of the current hop, selects an instance of the SF to be executed at the next hop, and tunnels the packet to the SFF for that SFI.

- If the current hop requires an {SFP, SI} and the next hop requires a {context label, SFI label}, it pops the {SFP, SI} from the top of the MPLS label stack and tunnels the packet to the SFF indicated by the context label.

- If the current hop requires a {context label, SFI label}, it pops the {context label, SFI label} from the top of the MPLS label stack.

    * If the new top of the MPLS label stack contains an {SFP, SI} label pair, it selects an SFI to use at the next hop, and tunnels the packet to SFF for that SFI.

* If the top of the MPLS label stack contains a {context label, SFI label}, it tunnels the packet to the SFF indicated by the context label.

8. A Note on Service Function Capabilities and SFC Proxies

The concept of an "SFC Proxy" is introduced in [RFC7665]. An SFC Proxy is logically located between an SFF and an SFI that is not "SFC-aware". Such SFIs are not capable of handling the SFC encapsulation (whether that be NSH or MPLS) and need the encapsulation stripped from the packets they are to process. In many cases, legacy SFIs that were once deployed as "bumps in the wire" fit into this category until they have been upgraded to be SFC-aware.

The job of an SFC Proxy is to remove and then reimpose SFC encapsulation so that the SFF is able to process as though it was communication with an SFC-aware SFI, and so that the SFI is unaware of the SFC encapsulation. In this regard, the job of an SFC Proxy is no different when NSH encapsulation is used and when MPLS encapsulation is used as described in this document, although (of course) it is different encapsulation bytes that must be removed and reimposed.

It should be noted that the SFC Proxy is a logical function. It could be implemented as a separate physical component on the path from the SFF to SFI, but it could be coresident with the SFF or it could be a component of the SFI. This is purely an implementation choice.

Note also that the delivery of metadata (see Section 11) requires specific processing if an SFC Proxy is in use. This is also no different when NSH or the MPLS encoding defined in this document is in use, and how it is handled will depend on how (or if) each non-SFC-aware SFI can receive metadata.

9. Control Plane Considerations

In order that a packet may be forwarded along an SFP several functional elements must be executed.

o Discovery/advertisement of SFIs.

o Computation of SFP.

o Programming of Classifiers.

o Advertisement of forwarding instructions.

Various approaches may be taken. These include a fully centralized model where SFFs report to a central controller the SFIs that they support, the central controller computes the SFP and programs the Classifiers, and (if the label swapping approach is taken) the central controller installs forwarding state in the SFFs that lie on the SFP.

Alternatively, a dynamic control plane may be used such as that described in [I-D.ietf-bess-nsh-bgp-control-plane]. In this case the SFFs use the control plane to advertise the SFIs that they support, a central controller computes the SFP and programs the Classifiers, and (if the label swapping approach is taken) the central controller uses the control plane to advertise the SFPs so that SFFs that lie on the SFP can install the necessary forwarding state.

10. Use of the Entropy Label

Entropy is used in ECMP situations to ensure that packets from the same flow travel down the same path, thus avoiding jitter or re-ordering issues within a flow.

Entropy is often determined by hashing on specific fields in a packet header such as the "five-tuple" in the IP and transport headers. However, when an MPLS label stack is present, the depth of the stack could be too large for some processors to correctly determine the entropy hash. This problem is addressed by the inclusion of an Entropy Label as described in [RFC6790].

When entropy is desired for packets as they are carried in MPLS tunnels over the underlay network, it is RECOMMENDED that an Entropy Label is included in the label stack immediately after the tunnel labels and before the SFC labels as shown in Figure 2 and Figure 3.

If an Entropy Label is present in an MPLS payload, it is RECOMMENDED that the initial Classifier use that value in an Entropy Label inserted in the label stack when the packet is forwarded (on the first tunnel) to the first SFF. In this case it is not necessary to remove the Entropy Label from the payload.

11. Metadata

Metadata is defined in [RFC7665] as providing "the ability to exchange context information between classifiers and SFs, and among SFs." [RFC8300] defines how this context information can be directly encoded in fields that form part of the NSH encapsulation.

The next two sections describe how metadata is associated with user data packets, and how metadata may be exchanged between SFC nodes in the network, when using an MPLS encoding of the logical representation of the NSH.

It should be noted that the MPLS encoding is slightly less functional than the direct use of the NSH. Both methods support metadata that is "per-SFP" or "per-packet-flow" (see [I-D.farrel-sfc-convent] for definitions of these terms), but "per-packet" metadata (where the metadata must be carried on each packet because it differs from one packet to the next even on the same flow or SFP) is only supported using the NSH and not using the mechanisms defined in this document.

11.1. Indicating Metadata in User Data Packets

Metadata is achieved in the MPLS realization of the logical NSH by the use of an SFC Metadata Label which uses the Extended Special Purpose Label construct [RFC7274]. Thus, three label stack entries are present as shown in Figure 4:

o The Extension Label (value 15)

o An extended special purpose label called the Metadata Label Indicator (MLI) (value TBD1 by IANA)

o The Metadata Label (ML).

```
 ----------------
| Extension = 15 |
+----------------+
|      MLI       |
+----------------+
| Metadata Label |
 ----------------
```

Figure 4: The MPLS SFC Metadata Label

The Metadata Label value is an index into a table of metadata that is programmed into the network using in-band or out-of-band mechanisms. Out-of-band mechanisms potentially include management plane and control plane solutions (such as [I-D.ietf-bess-nsh-bgp-control-plane]), but are out of scope for this document. The in-band mechanism is described in Section 11.2

The SFC Metadata Label (as a set of three labels as indicated in Figure 4) may be present zero, one, or more times in an MPLS SFC packet. For MPLS label swapping, the SFC Metadata Labels are placed immediately after the basic unit of MPLS label stack for SFC as shown in Figure 5. For MPLS label stacking, the SFC Metadata Labels can be present zero, one, or more times and are placed at the bottom of the label stack as shown in Figure 6.

```
  ----------------
 ~ Tunnel Labels  ~
 +----------------+
 ~   Optional     ~
 ~ Entropy Label  ~
 +----------------+
 |   SPI Label    |
 +----------------+
 |   SI Label     |
 +----------------+
 | Extension = 15 |
 +----------------+
 |     MLI        |
 +----------------+
 | Metadata Label |
 +----------------+
 ~     Other      ~
 |    Metadata    |
 ~  Label Triples ~
 +----------------+
 |                |
 ~    Payload     ~
 |                |
  ----------------
```

Figure 5: The MPLS SFC Label Stack for Label Swapping with Metadata Label

```
        -------------------
       ~   Tunnel Labels   ~
       +-------------------+
       ~     Optional      ~
       ~   Entropy Label   ~
       +-------------------+
       | SFC Context Label |
       +-------------------+
       |     SF Label      |
       +-------------------+
       ~                   ~
       +-------------------+
       | SFC Context Label |
       +-------------------+
       |     SF Label      |
       +-------------------+
       |   Extension = 15  |
       +-------------------+
       |        MLI        |
       +-------------------+
       |  Metadata Label   |
       +-------------------+
       ~       Other       ~
       |      Metadata     |
       ~   Label Triples   ~
       +-------------------+
       |                   |
       ~    Payload        ~
       |                   |
        -------------------
```

Figure 6: The MPLS SFC Label Stack for Label Stacking with Metadata Label

11.2. Inband Programming of Metadata

A mechanism for sending metadata associated with an SFP without a payload packet is described in [I-D.farrel-sfc-convent]. The same approach can be used in an MPLS network where the NSH is logically represented by an MPLS label stack.

The packet header is formed exactly as previously described in this document so that the packet will follow the SFP through the SFC network. However, instead of payload data, metadata is included after the bottom of the MPLS label stack. An Extended Special Purpose Label is used to indicate that the metadata is present. Thus, three label stack entries are present:

- o The Extension Label (value 15)
- o An extended special purpose label called the Metadata Present Indicator (MPI) (value TBD2 by IANA)
- o The Metadata Label (ML) that is associated with this metadata on this SFP and can be used to indicate the use of the metadata as described in Section 11.

The SFC Metadata Present Label, if present, is placed immediately after the last basic unit of MPLS label stack for SFC. The resultant label stacks are shown in Figure 7 for the MPLS label swapping case and Figure 8 for the MPLS label stacking case.

```
         ---------------
        ~ Tunnel Labels ~
        +---------------+
        ~   Optional    ~
        ~ Entropy Label ~
        +---------------+
        |   SPI Label   |
        +---------------+
        |   SI Label    |
        +---------------+
        | Extension = 15|
        +---------------+
        |      MPI      |
        +---------------+
        | Metadata Label|
        +---------------+
        |               |
        ~    Metadata   ~
        |               |
         ---------------
```

Figure 7: The MPLS SFC Label Stack for Label Swapping Carrying Metadata

```
      -------------------
     ~   Tunnel Labels   ~
     +-------------------+
     ~     Optional      ~
     ~   Entropy Label   ~
     +-------------------+
     | SFC Context Label |
     +-------------------+
     |     SF Label      |
     +-------------------+
     | SFC Context Label |
     +-------------------+
     |     SF Label      |
     +-------------------+
     ~                   ~
     +-------------------+
     | SFC Context Label |
     +-------------------+
     |     SF Label      |
     +-------------------+
     |   Extension = 15  |
     +-------------------+
     |        MPI        |
     +-------------------+
     |  Metadata Label   |
     +-------------------+
     |                   |
     ~    Metadata       ~
     |                   |
      -------------------
```

Figure 8: The MPLS SFC Label Stack for Label Stacking Carrying Metadata

In both cases the metadata is formatted as a TLV as shown in Figure 9.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |            Length             |         Metadata Type         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    ~                           Metadata                            ~
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9: The Metadata TLV

The fields of this TLV are interpreted as follows:

Length: The length of the metadata carried in the Metadata field in octets not including any padding.

Metadata Type: The type of the metadata present. Values for this field are taken from the "MD Types" registry maintained by IANA and defined in [RFC8300].

Metadata: The actual metadata formatted as described in whatever document defines the metadata. This field is end-padded with zero to three octets of zeroes to take it up to a four octet boundary.

12. Worked Examples

Consider the simplistic MPLS SFC overlay network shown in Figure 10. A packet is classified for an SFP that will see it pass through two Service Functions, SFa and SFb, that are accessed through Service Function Forwarders SFFa and SFFb respectively. The packet is ultimately delivered to destination, D.

Let us assume that the SFP is computed and assigned the SPI of 239. The forwarding details of the SFP are distributed (perhaps using the mechanisms of [I-D.ietf-bess-nsh-bgp-control-plane]) so that the SFFs are programmed with the necessary forwarding instructions.

The packet progresses as follows:

a. The Classifier assigns the packet to the SFP and imposes two label stack entries comprising a single basic unit of MPLS SFC representation:

   * The higher label stack entry contains a label carrying the SPI value of 239.

   * The lower label stack entry contains a label carrying the SI value of 255.

Further labels may be imposed to tunnel the packet from the Classifier to SFFa.

b. When the packet arrives at SFFa it strips any labels associated with the tunnel that runs from the Classifier to SFFa. SFFa examines the top labels and matches the SPI/SI to identify that the packet should be forwarded to SFa. The packet is forwarded to SFa unmodified.

c. SFa performs its designated function and returns the packet to SFFa.

d. SFFa modifies the SI in the lower label stack entry (to 254) and uses the SPI/SI to look up the forwarding instructions. It sends the packet with two label stack entries:

   * The higher label stack entry contains a label carrying the SPI value of 239.

   * The lower label stack entry contains a label carrying the SI value of 254.

   Further labels may be imposed to tunnel the packet from the SFFa to SFFb.

e. When the packet arrives at SFFb it strips any labels associated with the tunnel from SFFa. SFFb examines the top labels and matches the SPI/SI to identify that the packet should be forwarded to SFb. The packet is forwarded to SFb unmodified.

f. SFb performs its designated function and returns the packet to SFFb.

g. SFFb modifies the SI in the lower label stack entry (to 253) and uses the SPI/SI to lookup up the forwarding instructions. It determines that it is the last SFF in the SFP so it strips the two SFC label stack entries and forwards the payload toward D using the payload protocol.

```
        +---------------------------------------------------+
        |                   MPLS SFC Network                |
        |                                                   |
        |            +---------+       +---------+          |
        |            |   SFa   |       |   SFb   |          |
        |            +----+----+       +----+----+          |
        |               ^ | |             ^ | |             |
        |            (b)| | |(c)       (e)| | |(f)          |
        |       (a)     | | V     (d)     | | V    (g)      |
   +----------+ ---> +----+----+ ----> +----+----+ ---> +--------+
   |Classifier+------+  SFFa   +-------+  SFFb   +------+   D    |
   +----------+      +---------+       +---------+      +--------+
        |                                                   |
        +---------------------------------------------------+
```

Figure 10: Service Function Chaining in an MPLS Network

Alternatively, consider the MPLS SFC overlay network shown in Figure 11. A packet is classified for an SFP that will see it pass through two Service Functions, SFx and SFy, that are accessed through Service Function Forwarders SFFx and SFFy respectively. The packet is ultimately delivered to destination, D.

Let us assume that the SFP is computed and assigned the SPI of 239. However, the forwarding state for the SFP is not distributed and installed in the network. Instead it will be attached to the individual packets using the MPLS label stack.

The packet progresses as follows:

1. The Classifier assigns the packet to the SFP and imposes two basic units of MPLS SFC representation to describe the full SFP:

   * The top basic unit comprises two label stack entries as follows:

     + The higher label stack entry contains a label carrying the SFC context.

     + The lower label stack entry contains a label carrying the SF indicator for SFx.

   * The lower basic unit comprises two label stack entries as follows:

     + The higher label stack entry contains a label carrying the SFC context.

    + The lower label stack entry contains a label carrying the SF indicator for SFy.

   Further labels may be imposed to tunnel the packet from the Classifier to SFFx.

2. When the packet arrives at SFFx it strips any labels associated with the tunnel from the Classifier. SFFx examines the top labels and matches the context/SF values to identify that the packet should be forwarded to SFx. The packet is forwarded to SFx unmodified.

3. SFx performs its designated function and returns the packet to SFFx.

4. SFFx strips the top basic unit of MPLS SFC representation revealing the next basic unit. It then uses the revealed context/SF values to determine how to route the packet to the next SFF, SFFy. It sends the packet with just one basic unit of MPLS SFC representation comprising two label stack entries:

   * The higher label stack entry contains a label carrying the SFC context.

   * The lower label stack entry contains a label carrying the SF indicator for SFy.

   Further labels may be imposed to tunnel the packet from the SFFx to SFFy.

5. When the packet arrives at SFFy it strips any labels associated with the tunnel from SFFx. SFFy examines the top labels and matches the context/SF values to identify that the packet should be forwarded to SFy. The packet is forwarded to SFy unmodified.

6. SFy performs its designated function and returns the packet to SFFy.

7. SFFy strips the top basic unit of MPLS SFC representation revealing the payload packet. It forwards the payload toward D using the payload protocol.

```
        +---------------------------------------------------+
        |                   MPLS SFC Network                |
        |                                                   |
        |            +---------+       +---------+          |
        |            |   SFx   |       |   SFy   |          |
        |            +----+----+       +----+----+          |
        |               ^ | |             ^ | |             |
        |            (2)| | |(3)       (5)| | |(6)          |
        |       (1)     | | V     (4)     | | V    (7)      |
   +----------+ ---> +----+----+ ----> +----+----+ ---> +-------+
   |Classifier+------+  SFFx   +-------+  SFFy   +------+   D   |
   +----------+      +---------+       +---------+      +-------+
        |                                                   |
        +---------------------------------------------------+
```

Figure 11: Service Function Chaining Using MPLS Label Stacking

13. Security Considerations

Discussion of the security properties of SFC networks can be found in [RFC7665]. Further security discussion for the NSH and its use is present in [RFC8300].

It is fundamental to the SFC design that the classifier is a trusted resource which determines the processing that the packet will be subject to, including for example the firewall. It is also fundamental to the MPLS design that packets are routed through the network using the path specified by the node imposing the labels, and that labels are swapped or popped correctly. Where an SF is not encapsulation aware the encapsulation may be stripped by an SFC proxy such that packet may exist as a native packet (perhaps IP) on the path between SFC proxy and SF, however this is an intrinsic part of the SFC design which needs to define how a packet is protected in that environment.

Additionally, where a tunnel is used to link two non-MPLS domains, the tunnel design needs to specify how the tunnel is secured.

Thus the security vulnerabilities are addressed (or should be addressed) in all the underlying technologies used by this design, which itself does not introduce any new security vulnerabilities.

14. IANA Considerations

This document requests IANA to make allocations from the "Extended Special-Purpose MPLS Label Values" subregistry of the "Special-

Purpose Multiprotocol Label Switching (MPLS) Label Values" registry as follows:

Value	Description	
TBD1	Metadata Label Indicator (MLI)	[This.I-D]
TBD2	Metadata Present Indicator (MPI)	[This.I-D]

15. Acknowledgements

This document derives ideas and text from [I-D.ietf-bess-nsh-bgp-control-plane].

The authors are grateful to all those who contributed to the discussions that led to this work: Loa Andersson, Andrew G. Malis, Alexander Vainshtein, Joel M. Halpern, Tony Przygienda, Stuart Mackie, Keyur Patel, and Jim Guichard. Loa Andersson provided helpful review comments.

Thanks to Loa Andersson, Lizhong Jin, Matthew Bocci, and Mach Chen for reviews of this text.

16. References

16.1. Normative References

[I-D.farrel-sfc-convent]
Farrel, A. and J. Drake, "Operating the Network Service Header (NSH) with Next Protocol "None"", draft-farrel-sfc-convent-06 (work in progress), February 2018.

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

[RFC7274] Kompella, K., Andersson, L., and A. Farrel, "Allocating and Retiring Special-Purpose MPLS Labels", RFC 7274, DOI 10.17487/RFC7274, June 2014, <https://www.rfc-editor.org/info/rfc7274>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <https://www.rfc-editor.org/info/rfc8300>.

16.2. Informative References

[I-D.ietf-bess-nsh-bgp-control-plane] Farrel, A., Drake, J., Rosen, E., Uttaro, J., and L. Jalil, "BGP Control Plane for NSH SFC", draft-ietf-bess-nsh-bgp-control-plane-03 (work in progress), March 2018.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <https://www.rfc-editor.org/info/rfc3031>.

[RFC6790] Kompella, K., Drake, J., Amante, S., Henderickx, W., and L. Yong, "The Use of Entropy Labels in MPLS Forwarding", RFC 6790, DOI 10.17487/RFC6790, November 2012, <https://www.rfc-editor.org/info/rfc6790>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

Authors' Addresses

Adrian Farrel
Juniper Networks

Email: afarrel@juniper.net

Stewart Bryant
Huawei

Email: stewart.bryant@gmail.com

John Drake
Juniper Networks

Email: jdrake@juniper.net

SPRING WG S. Hegde
Internet-Draft Juniper Networks, Inc.
Intended status: Standards Track October 17, 2018
Expires: April 20, 2019

Traffic Accounting for MPLS Segment Routing Paths
draft-hegde-spring-traffic-accounting-for-sr-paths-02

Abstract

Traffic statistics form an important part of operations and maintenance data that are used to create demand matrices and for capacity planning in networks. Segment Routing (SR) is a source routing paradigm that uses stack of labels to represent a path. The SR path specific state is not stored in any other node in the network except the head-end node of the SR path. Traffic statistics specific to each SR path are an important component of the data which helps the controllers to lay out the SR paths in a way that optimizes the use of network resources. SR paths are inherently ECMP aware.

As SR paths do not have state in the core of the network, it is not possible to collect the SR path traffic statistics accurately on each interface. This document describes an MPLS forwarding plane mechanism to identify the SR path to which a packet belongs and so facilitate accounting of traffic for MPLS SR paths.

The mechanisms described in this document may also be applied to other MPLS paths (i.e., Label Switched Paths) and can be used to track traffic statistics in multipoint-to-point environments such as those where LDP is in use.

Requirements Language

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 RFC 2119 [RFC2119].

Status of This Memo

Table of Contents

1. Introduction

Figure 1 describes an SR enabled network with Node-SIDs and Anycast-SIDs assigned. The SR-Paths with label stacks are as shown in the diagram. The SR-Paths are created (possibly by a central controller) so as to maximize the network resource utilization such as bandwidth. Based on the traffic carried by the SR-Paths, they need to be re-routed occasionally to balance the bandwidth utilization. SR-Paths are inherently ECMP aware.

For example, SR-Path3 in the diagram is balanced across equal cost paths B->C->D and B->G->D. When there is congestion on the link between B and C, the SR path causing the congestion needs to be identified and re-routed. SR paths do not have separate control or forwarding state in any node other than the head-end. Traffic measurement at the head-end node is insufficient to determine the contribution of each SR path to the congestion on the link because of ECMP or Weighted ECMP balancing.

Per-SID traffic measurement on every interface gives some informtion about the traffic carried, but is not sufficient to correctly measure traffic carried by each SR path on the link. If it were possible to identify to which SR path each packet belonged, that information could be used by an external entity to re-route the SR paths to maximize resource utilization.

As SR paths do not have state in the core of the network, it is not possible to collect the SR path traffic statistics accurately on each interface. This document describes an MPLS forwarding plane mechanism to identify the SR path to which a packet belongs and so facilitate accounting of traffic for MPLS SR paths.

The mechanisms described in this document may also be applied to other MPLS paths (i.e., Label Switched Paths) and can be used to track traffic statistics in multipoint-to-point environments such as those where LDP is in use.

```
                         Anycast-SID:100
      SID:10    SID:20   SID:30          SID:40    SID:50
      +----+    +----+   +----+          +----+    +----+
      | A  |----| B  |---| C  |----------| D  |----| E  |
      +----+    +----+   +----+          +----+    +----+
                 /  \     |               /
                /    \    |              /
               /      \   |             /
          +----+      +----+           /
          | F  |      | G  |-----------
          +----+      +----+
          SID:60      SID:70
                      Anycast-SID:100

       SRGB: 1000-2000 on all routers
       SR-Path1: A-> 1020,1030
       SR-Path2: A-> 1020,1100,1040
       SR-Path3: F-> 1020,1040
       SR-Path4: A-> 1020,1040,1060
```

Figure 1: Sample Network

2. Motivation

The motivation of this document is to provide a solution to enable traffic measurement statistics per SR-Path on any node and any link in the network. The objectives listed below help to achieve the requirements in a variety of deployments.

1. The control plane MUST be free of any per SR path state.

2. The forwarding plane MUST be free of any per SR path state.

3. The number of counters created to measure traffic SHOULD be optimized.

4. The additional information carried in each packet SHOULD be minimized.

5. The mechanism SHOULD be applicable to all MPLS environments.

3. Terminology

Source-SID: The (globally unique) Node-SID of the head-end node which places traffic on the SR path. This is a 20 bit number excluding 0-15 and may be encoded in an MPLS label field.

SR-Path-Identifier: An SR-Path-Identifier is an identifier for each SR path in the network. It is unique within the scope of the node that allocated the identifier. If the identifier is allocated by the head-end node (the source) the combination of Source-SID and SR-Path Identifier uniquely identifies an SR path within a network. If the identifier is allocated by a central controller then the SR-Path Identifier is network unique. The SR-Path Identifier is a 19 bit number excluding the values 0-15 and may be encoded in an MPLS label field. See Section 4.

SR-Path-Indicator: The SR-Path-Indicator is an MPLS Special Purpose Label [RFC7274]. This label indicates the presence of an SR-Path Identifier and an Source Node-SID encoded in MPLS label stack entries and situated immediately below this label stack entry in the label stack.

SR-Path-Stats Labels: The SR-Path-Indicator, SR-Path-Identifier, and Source-SID together are termed as the SR-Path-Stats Labels.

4. SR-Path Identifier

4.1. Centrally Managed SR Paths

In controller-based deployments, a controller creates an SR policy, associates a segment list and a Binding SID to the policy, and sends it to the head-end of the SR path as described in [I-D.filsfils-spring-segment-routing-policy]. The controller may also allocate a network-unique SR-Path-Identifier and send it to the head-end along with the policy. When the head-end node receives this policy, if it has not been supplied with an SR-Path-Identifier, it creates a locally-unique identifier for each the SR path network and associates it with SR-TE Policy and advertizes it back to the controller using mechanisms described in [I-D.ietf-idr-te-lsp-distribution].

The SR-Path-Identifier is used for the purpose of traffic accounting as described in Section 5.

4.2. Locally Managed SR Paths

Deployments which do not use a central controller for managing the network configure locally manage SR-Paths on the head-end router. Every SR path in the network is identified using a Source-SID and a source-unique SR-Path-Identifier. The head-end node generates the SR-Path-Identifier for each SR path and associates it with the SR path. An Operator MAY also configure 19-bit globally unique Identifiers on each SR-Path and use it for accounting traffic as described in Section 5

5. Use of the SR-Path-Identifier and Source-SID

The SR-Path-Identifier is a 19 bit number created by the head-end node as described in Section 4. The SR-Path-Identifier and Source-SID are inserted in the packet below a Special Purpose Label called the SR-Path-Indicator. The three values are each carried in a label stack entry as shown in Figure 2.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          SR-Path-Indicator            | TC  |S|      TTL      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|        SR-Path-Identifier           | TC  |S|      TTL      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source-SID                   | TC  |S|      TTL      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: The SR-Path-Stats Labels Encoded in Label Stack Entries

The SR-Path-Indicator label value is TBD-1 to be assigned by IANA.

The SR-Path-Indicator label indicates that the MPLS label stack entries that follow carry an identifier of SR path. These label stack entries MUST NOT be used for forwarding, and if they are encountered at the top of the label stack (for example, at the egress node) they MUST be stripped.

The SR-Path-Identifier label stack entry is inserted immediately below the SR-Path-Indicator. The label field contains two elements:

- o The C-flag indicates whether the SR-Path-Identifier is allocated by a central controller or not. If the C-flag is set (one) then this indicates that the SR-Path-Identifier was allocated by a central controller and has global scope, and that a Source-SID is not included. If the C-flag is clear (zero) then the SR-Path-Identifier is scoped by the Source-SID that is included after the SR-Path-Identifier.

- o The SR-Path-Identifier identifies the SR path as described in Section 4.

The Source-SID is inserted immediately below the SR-Path-Identifier and is present only if indicated by the setting of the C-flag in the SR-Path-Identifier label stack entry. If present the Source-SID gives scope to the SR-Path-Identifier. The Source-SID is described in Section 4.

An intermediate node in the network can look into the packet and account the traffic based on the SR-Path-Identifier and Source-SID.

Because it is necessary that the SR-Path-Stats labels are removed when they are found at the top of the label stack, the node imposing the label stack (the ingress) must know which nodes are capable of stripping the labels. This ability is advertised in IGP advertisements defined in TBD and TBD.

6. Inserting the SR-Path-Identifier in Packets

The SR-Path-Identifier and Source-SID are used as a key to account the SR path traffic. The forwarding plane entities should look up the SR-Path-Identifier and Source-SID (if present) values to account the traffic against the right path counters.

The SR-Path-Stats Labels are normally placed at the bottom of the label stack.

Forwarding hardware may have limitations and not support accessing the label stack beyond certain depth. In such cases, the hardware will not be able to find the SR-Path-Stats Labels at the bottom of the label stack if the stack is too deep. To support traffic accounting in such cases it is necessary to insert the SR-Path-Stats Labels within the Readable Label Stack Depth Capability (RLDC) of the nodes in the SR path. The extensions defined in [I-D.ietf-ospf-segment-routing-msd] and [I-D.ietf-isis-segment-routing-msd] describe how the MSD supported by each node is advertised. The head-end node SHOULD insert the SR-Path-Stats Labels at a depth in the label stack such that the nodes in the SR path can access the SR-Path-Identifier for accounting. The SR-Path-Stats Labels may be present multiple times in the label stack of a packet.

In general, if all the nodes in the network support RLDC which is more than the label-stack depth being pushed at the head-end node then the SR-Path-Stats Labels SHOULD be pushed at the bottom of the label-stack. If there are service labels to be inserted, they MUST be pushed at the bottom of the stack. If entropy labels [RFC6790] are to be inserted they SHOULD be pushed next. The SR-Path-Stats Labels SHOULD be pushed next.

It is possible to partially deploy this feature when not all the nodes in the network support the extensions defined in this document. In such scenarios, the special labels MUST NOT get exposed on the top of the label stack at a node that does not support the extensions defined in this document. This may require multiple blocks of SR-Path-Stats Labels to be inserted in the packet header.

If the egress has not indicated that it is capable of removing the SR-Path-Stats Labels, then they MUST NOT be placed at the bottom of the label stack. In this case the SR-Path-Stats Labels SHOULD be placed at a point in the label stack such that they will be found at the top of stack by the latest node in the SR path that is capable of removing them. In this way, traffic accounting can be performed along as much of the SR path as possible.

7. Traffic-Accounting for Sub SR-Paths in the Network

SR paths may require large label stacks. Some hardware platforms do not support creating such large label stacks (i.e., imposing a large number of labels at once). To overcome this limitation sub-paths are created within the network, and Binding-SIDs are allocated to these sub-paths. When the label representing a Binding-SID is processed it is swapped for a stack of labels. When a head-end node builds the label stack for an SR path, it may use these Binding-SIDs to reduce the depth of the label stack it has to impose and effectively constructs the end-to-end SR path from a series of sub-paths

The sub-paths are not accounted separately. Accounting is performed on the end-to-end SR paths. However, edge routers MAY create Binding-SIDs for BGP-SR-TE Policies as described in [I-D.ietf-idr-segment-routing-te-policy]. Traffic accounting for the traffic carried on the SR paths indicated by these Binding-SIDs can be done separately by allocating separate SR-Path-Identifiers for these sub-paths.

8. Forwarding Plane Procedures

To support per-path traffic accounting, the forwarding plane in a router MUST look through the label stack of a packet for the first instance of the SR-Path-Indicator. The label value in the next label stack entry is the SR-Path-Identifierand the C-flag indicates whether a Source-SID label stack entry is also present. The label values are used as the key for accounting SR path traffic. If the Source-SID label stack entry is absent, an implementation may find it helpful to use a mock Source-SID value of zero for accounting purposes.

The SR-Path-Identifier may be located at different depth in the packet based on the RLDC of nodes in the network as described in Section 6. Finding the SR-Path-Identifier in the packet may be a costly operation and MUST NOT be done unless if SR path accounting is enabled on the device. Implementations MUST include a device-wide configuration option to enable and disable SR path accounting, and this option MUST default to "off". Implementations SHOULD include more granular configuration (such as per-interface).

A further configuration option is to limit the type of packets to which the procedures described in this section are applied. Thus, the forwarding plane could be configured to inspect only SR packets, or only MPLS packets established using a specific control plane technique (such as LDP). The top label on the incoming packet can be used to determine the nature of the packet and whether to search for the SR-Path-Identifier. The SR labels are predictable and are mostly assigned from SRGB or SRLB. If the top label belongs to any of these label blocks the procedures described in this section may be applied. If the SR label is allocated dynamically as in case of dynamic Adjacency-SIDs, it may be difficult to identify whether the label belongs to SR. It is RECOMMENDED to use configured Adjacency-SIDs when SR path traffic accounting is enabled.

If the top label of the incoming packet is of the right type for accounting and if other appropriate configuration options are enabled, then packet's label stack MUST be examined label by label until an SR-Path-Indicator label is found. The label below SR-Path-Indicator label is the SR-Path-Identifier label and the Source-SID label follows according to the setting of the C-flag. The {incoming interface, SR-Path-Identifier, Source SID} together are the key for traffic accounting. If the Source-SID label stack entry is absent, an implementation may find it helpful to use a mock Source-SID value of zero for accounting purposes.

If a counter does not already exist for that three-tuple, a new counter SHOULD be created. If a counter already exists, it MUST be incremented.

There is no requirement to preemptively create counters for every incoming interface and every SID: the counters need only be created, when a packet is received with the new SR-Path-identifier. This will significantly reduce the number of counters that need to be instantiated as not every interface will receive traffic for any particular SR path.

If the SR-Path-Indicator is the top label in a packet, the SR-Path-Stats labels are popped and further processing is based on the remaining labels in the label stack. Implementations MUST make sure the traffic accounting is carried out before the SR-Path-Stats labels are popped.

9. Consideration of Protection Mechanisms

SR paths typically consist of one or more Node-SIDs, Adjacency-SIDs, Anycast-SIDs, and Binding-SIDs. A variety of protection mechanisms may be in place for these SIDs as described in [I-D.ietf-spring-resiliency-use-cases]. When the head-end node inserts the SR-Path-Stats labels in the label stack, the place in the stack is decided based on whether the node where the special label gets exposed is capable of popping those labels.

When link protection is enabled, the traffic reaches the next-hop node before moving to towards the destination. With link-protection enabled, there is no risk of exposing the special labels at a node that does not support the extensions.

When node-protection is enabled, the traffic skips the next-hop node and reaches the next-next-hop towards the destination. In this case there is a possibility of special labels getting exposed at a node (the Merge Point) that does not support the extensions described in this document. In such cases, the node that receives the packet with special label at the top will discard the packet according to the processing rules of Section 3.18 of [RFC3031]. When using extensions described in this document for traffic accounting and with node-protection enabled in the network, it is RECOMMENDED to make sure all the nodes in the network support the extension.

10. Backward Compatibility

The extensions described in this document are backward compatible. Nodes that do not support the extensions defined in this document will not account the traffic (they will not search for the SR-Path-Indicator), but will forward traffic as normal.

While inserting the SR-Path-Stats labels, the head-end router MUST ensure that the labels are not exposed to the nodes that do not support them. If an error is made such that the SR-Path-Stats labels are exposed at the top of the label stack at a node that does not support this document then that node will discard the packets according to [RFC3031]. While the packets will be black-holed, no further harm will be caused to the network, and since this is a configuration or implementation error, this is an acceptable situation.

If an appropriate point in the label stack cannot be found for the insertion of the SR-Path-Stats labels, the head-end node, head-end MUST NOT insert the SR-Path-Stats labels, but SHOULD continue to label and transmit data. Under such circumstances the head-end node

SHOULD also log the event. A head-end or central controller MAY seek an alternate SR path that allows traffic accounting.

11. Scalability Considerations

The counter space is a limited resource in hardware. As described in Section 8 counters need only be created, when a packet is received with the an SR-Path-Identifier. Furthermore, counters need only be maintained where collection of statistics is configured.

Head-end nodes MUST NOT insert SR-Path-Stats labels by default. Careful configuration of which SR paths have statistics collection enabled will help to minimize the number of counters that need to be maintained at transit nodes.

Transit nodes that are constrained for the number of counters that they can support MAY implement mechanisms that sacrifice some under-used counters to create new counters.

As previously noted, the label stack is a prescious resource itself. That means that under some circumstances it is desirable to only use two labels in the SR-Path-Stats label sequence rather than three. This can be achieved by using a central controller to allocate SR-Path-Identifier values and set the C-flag to indicate that no Source-SID is used.

Conversely, in a large network with a central controller the SR-Path-Identifier may be a prescious resource. That is, there may be more than 2^19 SR paths that need identifiers to be allocated. In this case, a central controller may use knowledge of label stack depth and network node capabilities to allocate SR-Path-Indicators that include a Source-SID (set to indicate the controller, itself) where that would not cause a problem in the network.

12. Security Considerations

As noted in Section 11 the counter space is a limited resource in hardware. This document introduces dynamic creation of counters based on packet headers of the incoming packets. There is the possibility that a DOS attack is mounted by requesting new counter creation on each packet. Implementations SHOULD monitor the counter space and generate appropriate warnings if the counter space is getting exhausted. Implementations SHOULD control the rate at which the counters get created to mitigate DOS attacks.

13. IANA Considerations

IANA maintains a registry called the "Multiprotocol Label Switching Architecture (MPLS) Label Values" registry. IANA is requested to make a new assignment from this registry as follows:

Value	Description	Reference
TBD-1	SR Path Indicator	[This.I-D]

14. Acknowledgements

Thanks to John Drake, Harish Sitaraman, and Ron Bonica for helpful discussions.

15. Contributors

Adrian Farrel
Juniper Networks

Email: afarrel@juinper.net

16. References

16.1. Normative References

[I-D.ietf-idr-te-lsp-distribution]
Previdi, S., Talaulikar, K., Dong, J., Chen, M., Gredler, H., and J. Tantsura, "Distribution of Traffic Engineering (TE) Policies and State using BGP-LS", draft-ietf-idr-te-lsp-distribution-09 (work in progress), June 2018.

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

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <https://www.rfc-editor.org/info/rfc3031>.

16.2. Informative References

[I-D.filsfils-spring-segment-routing-policy]
Filsfils, C., Sivabalan, S., Hegde, S., daniel.voyer@bell.ca, d., Lin, S., bogdanov@google.com, b., Krol, P., Horneffer, M., Steinberg, D., Decraene, B., Litkowski, S., Mattes, P., Ali, Z., Talaulikar, K., Liste, J., Clad, F., and K. Raza, "Segment Routing Policy Architecture", draft-filsfils-spring-segment-routing-policy-06 (work in progress), May 2018.

[I-D.ietf-idr-segment-routing-te-policy]
Previdi, S., Filsfils, C., Jain, D., Mattes, P., Rosen, E., and S. Lin, "Advertising Segment Routing Policies in BGP", draft-ietf-idr-segment-routing-te-policy-04 (work in progress), July 2018.

[I-D.ietf-isis-segment-routing-msd]
Tantsura, J., Chunduri, U., Aldrin, S., and L. Ginsberg, "Signaling MSD (Maximum SID Depth) using IS-IS", draft-ietf-isis-segment-routing-msd-19 (work in progress), October 2018.

[I-D.ietf-ospf-segment-routing-msd]
Tantsura, J., Chunduri, U., Aldrin, S., and P. Psenak, "Signaling MSD (Maximum SID Depth) using OSPF", draft-ietf-ospf-segment-routing-msd-23 (work in progress), October 2018.

[I-D.ietf-spring-resiliency-use-cases]
Filsfils, C., Previdi, S., Decraene, B., and R. Shakir, "Resiliency use cases in SPRING networks", draft-ietf-spring-resiliency-use-cases-12 (work in progress), December 2017.

[RFC6790] Kompella, K., Drake, J., Amante, S., Henderickx, W., and L. Yong, "The Use of Entropy Labels in MPLS Forwarding", RFC 6790, DOI 10.17487/RFC6790, November 2012, <https://www.rfc-editor.org/info/rfc6790>.

[RFC7274] Kompella, K., Andersson, L., and A. Farrel, "Allocating and Retiring Special-Purpose MPLS Labels", RFC 7274, DOI 10.17487/RFC7274, June 2014, <https://www.rfc-editor.org/info/rfc7274>.

Author's Address

Shraddha Hegde
Juniper Networks, Inc.
Embassy Business Park
Bangalore, KA 560093
India

Email: shraddha@juniper.net

MPLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 18, 2018

K. Raza
R. Asati
Cisco Systems, Inc.

X. Liu
Jabil

S. Esale
Juniper Networks

X. Chen
Huawei Technologies

H. Shah
Ciena Corporation

September 14, 2017

YANG Data Model for MPLS LDP
draft-ietf-mpls-ldp-yang-02

Abstract

This document describes a YANG data model for Multi-Protocol Label Switching (MPLS) Label Distribution Protocol (LDP). This model also serves as the base model that is augmented to define Multipoint LDP (mLDP) model.

Status of This Memo

Table of Contents

1. Introduction

The Network Configuration Protocol (NETCONF) [RFC6241] is one of the network management protocols that defines mechanisms to manage network devices. YANG [RFC6020] is a modular language that represents data structures in an XML tree format, and is used as a data modelling language for the NETCONF.

This document introduces a YANG data model for MPLS Label Distribution Protocol (LDP) [RFC5036]. This model also covers LDP IPv6 [RFC7552] and LDP capabilities [RFC5561].

The data model is defined for following constructs that are used for managing the protocol:

o Configuration

o Operational State

o Executables (Actions)

o Notifications

This document is organized to define the data model for each of the above constructs in the sequence as listed above.

1.1. Base and Extended

The configuration and state items are divided into following two broad categories:

o Base

o Extended

The "base" category contains the basic and fundamental features that are covered in LDP base specification [RFC5036] and constitute the minumum requirements for a typical base LDP deployment. Whereas, the "extended" category contains all other non-base features. All the items in a base category are mandatory and hence no "if-feature" is allowed under the "base" category model. The base and extended catogories are defined in their own modules as described later.

The example of base feature includes the configuration of LDP lsr-id, enabling LDP interfaces, setting password for LDP session etc., whereas the examples of extended feature include inbound/outbound label policies, igp sync, downstream-on-demand etc. This is worth higlighting that LDP IPv6 [RFC7552] is also categorized as an extended feature.

While "base" model support will suffice for small deployments, it is expected that large deployments will require not only the "base" module support from the vendors but also the support for "extended" model for some extended feature(s) of interest.

2. Specification of Requirements

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].

In this document, the word "IP" is used to refer to both IPv4 and IPv6, unless otherwise explicitly stated. For example, "IP address family" means and be read as "IPv4 and/or IPv6 address family"

3. Overview

This document defines two new modules for LDP YANG support:

o "ietf-mpls-ldp" module that models the base LDP features and augments /rt:routing/rt:control-plane-protocols defined in [RFC8022].

o "ietf-mpls-ldp-extended" module that models the extended LDP features and augments the base LDP.

It is to be noted that mLDP data model [I-D.ietf-mpls-mldp-yang] augments LDP base and extended models to model the base and extended mLDP features respectively.

There are four main containers in our module(s):

o Read-Write parameters for configuration (Discussed in Section 4)

o Read-only parameters for operational state (Discussed in Section 5)

o Notifications for events (Discussed in Section 6)

o RPCs for executing commands to perform some action (Discussed in Section 7)

For the configuration and state data, this model follows the similar approach described in [I-D.openconfig-netmod-opstate] to represent the configuration (intended state) and operational (applied and derived) state. This means that for every configuration (rw) item, there is an associated (ro) item under "state" container to represent the applied state. Furthermore, protocol derived state is also kept under "state" tree corresponding to the protocol area (discovery, peer etc.). [Ed note: This document will be (re-)aligned with [I-D.openconfig-netmod-opstate] once that specification is adopted as a WG document].

Following diagram depicts high level LDP yang tree organization and hierarchy:

```
       module: ietf-mpls-ldp
            +-- rw routing
              +-- rw control-plane-protocols
                 +-- rw mpls-ldp
                     +-- rw global
                     |    +-- rw config
                     |    |    +-- rw ...             // base
                     |    |    +-- rw ldp-ext: ....   // extended
                     |    |    ...
                     |    +-- ro state
                     |         +-- ro ...             // base
                     |    |    +-- ro ldp-ext: ....   // extended
                     |         ...
                     +-- rw ...
                     |    +-- rw config
                     |    |    +-- rw ...             // base
                     |    |    +-- rw ldp-ext: ....   // extended
                     |    |    ...
                     |    +-- ro state
                     |         +-- ro ...             // base
                     |    |    +-- ro ldp-ext: ....   // extended
                     |         ...
                     +-- rw ...
                     ...

       rpcs:
          +-- x mpls-ldp-some_action
          +-- x . . . . .

       notifications:
          +--- n mpls-ldp-some_event
          +--- n ...
```

Figure 1

Before going into data model details, it is important to take note of the following points:

- o This module aims to address only the core LDP parameters as per RFC specification, as well as some widely deployed non-RFC features (such as label policies, session authentication etc). Any vendor specific feature should be defined in a vendor-specific augmentation of this model.

- o Multi-topology LDP [RFC7307] is beyond the scope of this document.

- o This module does not cover any applications running on top of LDP, nor does it cover any OAM procedures for LDP.

- o This model is a VPN Forwarding and Routing (VRF)-centric model. It is important to note that [RFC4364] defines VRF tables and default forwarding tables as different, however from a yang modelling perspective this introduces unnecessary complications, hence we are treating the default forwarding table as just another VRF.

- o A "network-instance", as defined in [I-D.rtgyangdt-rtgwg-ni-model], refers to a VRF instance (both default and non-default) within the scope of this model.

- o This model supports two address-families, namely "ipv4" and "ipv6".

- o This model assumes platform-wide label space (i.e. label space Id of zero). However, when Upstream Label assignment [RFC6389] is in use, an upstream assigned label is looked up in a Context-Specific label space as defined in [RFC5331].

- o The label and peer policies (including filters) are defined using a prefix-list. When used for a peer policy, the prefix refers to the LSR Id of the peer. The prefix-list is referenced from routing-policy model as defined in [I-D.ietf-rtgwg-policy-model].

- o This model uses the terms LDP "neighbor"/"adjacency", "session", and "peer" with the following semantics:

  * Neighbor/Adjacency: An LDP enabled LSR that is discovered through LDP discovery mechanisms.

  * Session: An LDP neighbor with whom a TCP connection has been established.

  * Peer: An LDP session which has successfully progressed beyond its initialization phase and is either already exchanging the bindings or is ready to do so.

  It is to be noted that LDP Graceful Restart mechanisms defined in [RFC3478] allow keeping the exchanged bindings for some time after a session goes down with a peer. We call such a state belonging to a "stale" peer -- i.e. keeping peer bindings from a peer with whom currently there is either no connection established or connection is established but GR session is in recovery state. When used in this document, the above terms will refer strictly to the semantics and definitions defined for them.

A graphical representation of LDP YANG data model is presented in Figure 4, Figure 5, Figure 7, Figure 8, Figure 14, and Figure 15. Whereas, the actual model definition in YANG is captured in Section 9.

While presenting the YANG tree view and actual .yang specification, this document assumes readers' familiarity with the concepts of YANG modeling, its presentation and its compilation.

4. Configuration

This specification defines the configuration parameters for base LDP as specified in [RFC5036] and LDP IPv6 [RFC7552]. Moreover, it incorporates provisions to enable LDP Capabilities [RFC5561], and defines some of the most significant and commonly used capabilities such as Typed Wildcard FEC [RFC5918], End-of-LIB [RFC5919], and LDP Upstream Label Assignment [RFC6389].

This model augments /rt:routing/rt:control-plane-protocols that is defined in [RFC8022]. For LDP interfaces, this model refers the MPLS interface as defined under MPLS base specification [I-D.ietf-mpls-base-yang]. Furthermore, as mentioned earlier, the configuration tree presents read-write intended configuration leave/ items as well as read-only state of the applied configuration. The former is listed under "config" container and latter under "state" container.

Following is the high-level configuration organization for base LDP:

```
        augment /rt:routing/rt:control-plane-protocols/rt:control-plane-protocol
:
              +-- mpls-ldp
                  +-- global
                  |     +-- ...
                  |     +-- ...
                  |     +-- address-families
                  |     |    +-- ipv4
                  |     |      +-- . . .
                  |     |      +-- . . .
                  |     |      +-- label-policy
                  |     |        +-- ...
                  |     |        +-- ...
                  |     +-- capability
                  |     |    +-- ...
                  |     |    +-- ...
                  |     +-- discovery
                  |          +-- interfaces
                  |          |    +-- ...
                  |          |    +-- ...
                  |          |    +-- interface* [interface]
                  |          |      +-- ...
                  |          |      +-- address-families
                  |          |        +-- ipv4
                  |          |          +-- ...
                  |          |          +-- ...
                  |          +-- targeteted
                  |              +-- ...
                  |              +-- address-families
                  |                +-- ipv4
                  |                  +- target* [adjacent-address]
                  |                    +- ...
                  |                    +- ...
                  +-- peers
                       +-- ...
                       +-- ...
                       +-- peer*
                           +-- ...
                           +-- ...
```

Figure 2

Following is the high-level configuration organization for extended LDP:

```
        augment /rt:routing/rt:control-plane-protocols/rt:control-plane-protocol
:
              +-- mpls-ldp
```

```
                  +-- global
                  |     +-- ...
                  |     +-- ...
                  |     +-- address-families
                  |     |     +-- ipv4
                  |     |     | +-- . . .
                  |     |     | +-- . . .
                  |     |     | +-- label-policy
                  |     |     |     +-- ...
                  |     |     |     +-- ...
                  |     |     +-- ipv6
                  |     |       +-- . . .
                  |     |       +-- . . .
                  |     |       +-- label-policy
                  |     |         +-- ...
                  |     |         +-- ...
                  |     +-- label-policy
                  |     |     +-- ...
                  |     |     +-- ...
                  |     +-- capability
                  |     |     +-- ...
                  |     |     +-- ...
                  |     +-- discovery
                  |           +-- interfaces
                  |           |     +-- ...
                  |           |     +-- ...
                  |           |     +-- interface* [interface]
                  |           |       +-- ...
                  |           |       +-- address-families
                  |           |         +-- ipv4
                  |           |         | +-- ...
                  |           |         | +-- ...
                  |           |         +-- ipv6
                  |           |           +-- ...
                  |           |           +-- ...
                  |           +-- targeteted
                  |               +-- ...
                  |               +-- address-families
                  |                 +-- ipv4
                  |                 | +- target* [adjacent-address]
                  |                 |   +- ...
                  |                 |   +- ...
                  |                 +-- ipv6
                  |                   +- target* [adjacent-address]
                  |                     +- ...
                  |                     +- ...
                  +-- forwarding-nexthop
                  |   +-- ...
                  |   +-- ...
                  +-- peers
                       +-- ...
                       +-- ...
                       +-- peer*
                           +-- ...
                           +-- ...
                           +-- label-policy
                           |   +-- ..
                           +-- address-families
                              +-- ipv4
                              |   +-- label-policies
                              |       +-- ...
                              +-- ipv6
                                  +-- label-policies
                                      +-- ...
```

Figure 3

Given the configuration hierarchy, the model allows inheritance such that an item in a child tree is able to derive value from a similar or related item in one of the parent. For instance, hello holdtime can be configured per-VRF or per-VRF-interface, thus allowing inheritance as well flexibility to override with a different value at any child level.

4.1. Configuration Tree

4.1.1. Base

Following is a simplified graphical representation of the data model for LDP base configuration

```
   module: ietf-mpls-ldp
   augment /rt:routing/rt:control-plane-protocols:
      +--rw mpls-ldp!
         +--rw global
         |  +--rw config
         |  |  +--rw capability
         |  |  +--rw graceful-restart
         |  |  |  +--rw enable?                boolean
         |  |  |  +--rw reconnect-time?        uint16
         |  |  |  +--rw recovery-time?         uint16
         |  |  |  +--rw forwarding-holdtime?   uint16
         |  |  +--rw lsr-id?            yang:dotted-quad
         |  +--rw address-families
         |  |  +--rw ipv4
         |  |     +--rw config
         |  |        +--rw enable?         boolean
         |  |        +--rw label-policy
         |  |           +--rw advertise
         |  |              +--rw egress-explicit-null
         |  |                 +--rw enable?   boolean
         |  +--rw discovery
         |     +--rw interfaces
         |     |  +--rw config
         |     |  |  +--rw hello-holdtime?   uint16
         |     |  |  +--rw hello-interval?   uint16
         |     |  +--rw interface* [interface]
         |     |     +--rw interface           mpls-interface-ref
         |     |     +--rw address-families
         |     |        +--rw ipv4
         |     |           +--rw config
         |     |              +--rw enable?   boolean
         |     +--rw targeted
         |        +--rw config
         |        |  +--rw hello-holdtime?   uint16
         |        |  +--rw hello-interval?   uint16
         |        |  +--rw hello-accept
         |        |     +--rw enable?   boolean
         |        +--rw address-families
         |           +--rw ipv4
         |              +--rw target* [adjacent-address]
         |                 +--rw adjacent-address    inet:ipv4-address
         |                 +--rw config
         |                    +--rw enable?          boolean
         |                    +--rw local-address?   inet:ipv4-address
         +--rw peers
            +--rw config
            |  +--rw authentication
            |  |  +--rw (auth-type-selection)?
            |  |     +--:(auth-key)
            |  |        +--rw md5-key?   string
            |  +--rw capability
            |  +--rw session-ka-holdtime?   uint16
            |  +--rw session-ka-interval?   uint16
            +--rw peer* [lsr-id]
               +--rw lsr-id    yang:dotted-quad
               +--rw config
                  +--rw authentication
                     +--rw (auth-type-selection)?
                        +--:(auth-key)
                           +--rw md5-key?   string
```

Figure 4

4.1.2. Extended

Following is a simplified graphical representation of the data model for LDP extended configuration

```
module: ietf-mpls-ldp
augment /rt:routing/rt:control-plane-protocols:
   +--rw mpls-ldp!
      +--rw global
      |  +--rw config
      |  |  +--rw capability
      |  |  |  +--rw ldp-ext:end-of-lib {capability-end-of-lib}?
      |  |  |  |  +--rw ldp-ext:enable?   boolean
      |  |  |  +--rw ldp-ext:typed-wildcard-fec {capability-typed-wildcard-fec}?
      |  |  |  |  +--rw ldp-ext:enable?   boolean
      |  |  |  +--rw ldp-ext:upstream-label-assignment {capability-upstream-labe
l-assignment}?
      |  |  |     +--rw ldp-ext:enable?   boolean
      |  |  +--rw graceful-restart
      |  |  |  +--rw ldp-ext:helper-enable?   boolean {graceful-restart-helper-m
ode}?
      |  |  +--rw ldp-ext:igp-synchronization-delay?   uint16
      |  |  +--rw ldp-ext:label-policy
      |  |     +--rw ldp-ext:advertise
      |  |        +--rw ldp-ext:egress-explicit-null
      |  |           +--rw ldp-ext:enable?   boolean
      |  +--rw address-families
      |  |  +--rw ipv4
      |  |  |  +--rw config
      |  |  |     +--rw label-policy
      |  |  |     |  +--rw advertise
      |  |  |     |  |  +--rw ldp-ext:prefix-list?    prefix-list-ref
      |  |  |     |  +--rw ldp-ext:accept
      |  |  |     |  |  +--rw ldp-ext:prefix-list?   prefix-list-ref
      |  |  |     |  +--rw ldp-ext:assign {policy-label-assignment-config}?
      |  |  |     |     +--rw ldp-ext:independent-mode
      |  |  |     |     |  +--rw ldp-ext:prefix-list?   prefix-list-ref
      |  |  |     |     +--rw ldp-ext:ordered-mode {policy-ordered-label-config}
?
      |  |  |     |        +--rw ldp-ext:egress-prefix-list?   prefix-list-ref
      |  |  |     +--rw ldp-ext:transport-address?   inet:ipv4-address
      |  |  +--rw ldp-ext:ipv6
      |  |     +--rw ldp-ext:config
      |  |        +--rw ldp-ext:enable?             boolean
      |  |        +--rw ldp-ext:label-policy
      |  |        |  +--rw ldp-ext:advertise
      |  |        |  |  +--rw ldp-ext:egress-explicit-null
      |  |        |  |  |  +--rw ldp-ext:enable?   boolean
      |  |        |  |  +--rw ldp-ext:prefix-list?            prefix-list-ref
      |  |        |  +--rw ldp-ext:accept
      |  |        |  |  +--rw ldp-ext:prefix-list?   prefix-list-ref
      |  |        |  +--rw ldp-ext:assign {policy-label-assignment-config}?
      |  |        |     +--rw ldp-ext:independent-mode
      |  |        |     |  +--rw ldp-ext:prefix-list?   prefix-list-ref
      |  |        |     +--rw ldp-ext:ordered-mode {policy-ordered-label-config}
?
      |  |        |        +--rw ldp-ext:egress-prefix-list?   prefix-list-ref
      |  |        +--rw ldp-ext:transport-address?   inet:ipv6-address
      |  +--rw discovery
      |  |  +--rw interfaces
      |  |  |  +--rw interface* [interface]
      |  |  |     +--rw interface           mpls-interface-ref
      |  |  |     +--rw address-families
      |  |  |     |  +--rw ipv4
      |  |  |     |  |  +--rw config
      |  |  |     |  |     +--rw ldp-ext:transport-address?   union
      |  |  |     |  +--rw ldp-ext:ipv6
      |  |  |     |     +--rw ldp-ext:config
      |  |  |     |        +--rw ldp-ext:enable?              boolean
      |  |  |     |        +--rw ldp-ext:transport-address?   union
      |  |  |     +--rw ldp-ext:config
      |  |  |        +--rw ldp-ext:hello-holdtime?              uint16
      |  |  |        +--rw ldp-ext:hello-interval?              uint16
      |  |  |        +--rw ldp-ext:igp-synchronization-delay?   uint16 {per-inte
rface-timer-config}?
      |  |  +--rw targeted
      |  |     +--rw config
      |  |     |  +--rw hello-accept
      |  |     |     +--rw ldp-ext:neighbor-list?   neighbor-list-ref {policy-ta
rgeted-discovery-config}?
      |  |     +--rw address-families
      |  |        +--rw ldp-ext:ipv6
      |  |           +--rw ldp-ext:target* [adjacent-address]
      |  |              +--rw ldp-ext:adjacent-address    inet:ipv6-address
      |  |              +--rw ldp-ext:config
      |  |                 +--rw ldp-ext:enable?          boolean
      |  |                 +--rw ldp-ext:local-address?   inet:ipv6-address
      |  +--rw ldp-ext:forwarding-nexthop {forwarding-nexthop-config}?
      |     +--rw ldp-ext:interfaces
      |        +--rw ldp-ext:interface* [interface]
      |           +--rw ldp-ext:interface         ldp:mpls-interface-ref
      |           +--rw ldp-ext:address-family* [afi]
      |              +--rw ldp-ext:afi       ldp:ldp-address-family
      |              +--rw ldp-ext:config
      |                 +--rw ldp-ext:ldp-disable?   boolean
      +--rw peers
         +--rw config
         |  +--rw authentication
         |  |  +--rw (auth-type-selection)?
         |  |     +--:(ldp-ext:auth-key-chain)
         |  |        +--rw ldp-ext:key-chain?   key-chain:key-chain-ref
         |  +--rw ldp-ext:session-downstream-on-demand {session-downstream-on-de
mand-config}?
         |     +--rw ldp-ext:enable?      boolean
         |     +--rw ldp-ext:peer-list?   peer-list-ref
         +--rw peer* [lsr-id]
            +--rw lsr-id    yang:dotted-quad
            +--rw config
               +--rw authentication
               |  +--rw (auth-type-selection)?
               |     +--:(ldp-ext:auth-key-chain)
               |        +--rw ldp-ext:key-chain?   key-chain:key-chain-ref
               +--rw ldp-ext:admin-down?            boolean
               +--rw ldp-ext:label-policy
               |  +--rw ldp-ext:advertise
               |  |  +--rw ldp-ext:prefix-list?   prefix-list-ref
               |  +--rw ldp-ext:accept
               |     +--rw ldp-ext:prefix-list?   prefix-list-ref
               +--rw ldp-ext:graceful-restart
               |  +--rw ldp-ext:enable?           boolean
               |  +--rw ldp-ext:reconnect-time?   uint16
               |  +--rw ldp-ext:recovery-time?    uint16
               +--rw ldp-ext:session-ka-holdtime?   uint16
               +--rw ldp-ext:session-ka-interval?   uint16
               +--rw ldp-ext:address-families
                  +--rw ldp-ext:ipv4
                  |  +--rw ldp-ext:label-policy
                  |     +--rw ldp-ext:advertise
                  |     |  +--rw ldp-ext:prefix-list?   prefix-list-ref
                  |     +--rw ldp-ext:accept
                  |        +--rw ldp-ext:prefix-list?   prefix-list-ref
                  +--rw ldp-ext:ipv6
                     +--rw ldp-ext:label-policy
                        +--rw ldp-ext:advertise
                        |  +--rw ldp-ext:prefix-list?   prefix-list-ref
                        +--rw ldp-ext:accept
                           +--rw ldp-ext:prefix-list?   prefix-list-ref
```

Figure 5

4.2. Configuration Hierarchy

The LDP configuration container is logically divided into following high-level config areas:

```
      Per-VRF parameters
          o Global parameters
          o Per-address-family parameters
          o LDP Capabilities parameters
          o Hello Discovery parameters
              - interfaces
                 - Per-interface:
                     Global
                     Per-address-family
              - targeted
                 - Per-target
         o Peer parameters
              - Global
              - Per-peer
                 Per-address-family
                 Capabilities parameters
         o Forwarding parameters
```

Figure 6

Following subsections briefly explain these configuration areas.

4.2.1. Per-VRF parameters

LDP module resides under an network-instance and the scope of any LDP configuration defined under this tree is per network-instance (per-VRF). This configuration is further divided into sub categories as follows.

4.2.1.1. Per-VRF global parameters

There are configuration items that are available directly under a VRF instance and do not fall under any other sub tree. Example of such a parameter is LDP LSR id that is typically configured per VRF. To keep legacy LDP features and applications working in an LDP IPv4 networks with this model, this document recommends an operator to pick a routable IPv4 unicast address as an LSR Id.

4.2.1.2. Per-VRF Capabilities parameters

This container falls under global tree and holds the LDP capabilities that are to be enabled for certain features. By default, an LDP capability is disabled unless explicitly enabled. These capabilities are typically used to negotiate with LDP peer(s) the support/non-support related to a feature and its parameters. The scope of a capability enabled under this container applies to all LDP peers in the given VRF instance. There is also a peer level capability container that is provided to override a capability that is enabled/ specified at VRF level.

4.2.1.3. Per-VRF Per-Address-Family parameters

Any LDP configuration parameter related to IP address family (AF) whose scope is VRF wide is configured under this tree. The examples of per-AF parameters include enabling LDP for an address family, prefix-list based label policies, and LDP transport address.

4.2.1.4. Per-VRF Hello Discovery parameters

This container is used to hold LDP configuration related to Hello and discovery process for both basic (link) and extended (targeted) discovery.

The "interfaces" is a container to configure parameters related to VRF interfaces. There are parameters that apply to all interfaces (such as hello timers), as well as parameters that can be configured per-interface. Hence, an interface list is defined under "interfaces" container. The model defines parameters to configure per-interface non AF related items, as well as per-interface per-AF items. The example of former is interface hello timers, and example of latter is enabling hellos for a given AF under an interface.

The "targeted" container under a VRF instance allows to configure LDP targeted discovery related parameters. Within this container, the "target" list provides a mean to configure multiple target addresses to perform extended discovery to a specific destination target, as well as to fine-tune the per-target parameters.

4.2.1.5. Per-VRF Peer parameters

This container is used to hold LDP configuration related to LDP sessions and peers under a VRF instance. This container allows to configure parameters that either apply on VRF's all peers or a subset (peer-list) of VRF peers. The example of such parameters include authentication password, session KA timers etc. Moreover, the model also allows per-peer parameter tuning by specifying a "peer" list under the "peers" container. A peer is uniquely identified using its LSR Id and hence LSR Id is the key for peer list

Like per-interface parameters, some per-peer parameters are AF-agnostic (i.e. either non AF related or apply to both IP address families), and some that belong to an AF. The example of former is per-peer session password configuration, whereas the example of latter is prefix-list based label policies (inbound and outbound) that apply to a given peer.

4.2.1.6. Per-VRF Forwarding parameters

This container is used to hold configuration used to control LDP forwarding behavior under a VRF instance. One example of a configuration under this container is when a user wishes to enable neighbor discovery on an interface but wishes to disable use of the same interface as forwarding nexthop. This example configuration makes sense only when there are more than one LDP enabled interfaces towards the neighbor.

5. Operational State

Operational state of LDP can be queried and obtained from read-only state containers that fall under the same tree (/rt:routing/ rt:control-plane-protocols/) as the configuration.

Please note this state tree refers both the configuration "applied" state as well as the "derived" state related to the protocol. [Ed note: This is where this model differs presently from [I-D.openconfig-netmod-opstate] and subject to alignment in later revisions]

5.1. Operational Tree

5.1.1. Base

Following is a simplified graphical representation of the base data model for LDP operational state.

```
 module: ietf-mpls-ldp
augment /rt:routing/rt:control-plane-protocols:
   +--rw mpls-ldp!
      +--rw global
      |  +--ro state
      |  |  +--ro capability
      |  |  +--ro graceful-restart
      |  |  |  +--ro enable?                boolean
      |  |  |  +--ro reconnect-time?        uint16
      |  |  |  +--ro recovery-time?         uint16
      |  |  |  +--ro forwarding-holdtime?   uint16
      |  |  +--ro lsr-id?            yang:dotted-quad
      |  +--rw address-families
      |  |  +--rw ipv4
      |  |     +--ro state
      |  |        +--ro enable?                             boolean
      |  |        +--ro label-distribution-controlmode?   enumeration
      |  |        +--ro label-policy
      |  |        |  +--ro advertise
      |  |        |     +--ro egress-explicit-null
      |  |        |        +--ro enable?   boolean
      |  |        +--ro bindings
      |  |           +--ro address* [address]
      |  |           |  +--ro address               inet:ipv4-address
      |  |           |  +--ro advertisement-type?   advertised-received
      |  |           |  +--ro peer?                 leafref
      |  |           +--ro fec-label* [fec]
      |  |              +--ro fec     inet:ipv4-prefix
      |  |              +--ro peer* [peer advertisement-type]
      |  |                 +--ro peer                  leafref
      |  |                 +--ro advertisement-type    advertised-received
      |  |                 +--ro label?                rt-types:mpls-label
      |  |                 +--ro used-in-forwarding?   boolean
      |  +--rw discovery
      |     +--rw interfaces
      |     |  +--ro state
      |     |  |  +--ro hello-holdtime?   uint16
      |     |  |  +--ro hello-interval?   uint16
      |     |  +--rw interface* [interface]
      |     |     +--rw interface           mpls-interface-ref
      |     |     +--ro state
      |     |     |  +--ro next-hello?   uint16
      |     |     +--rw address-families
      |     |        +--rw ipv4
      |     |           +--ro state
      |     |              +--ro enable?              boolean
      |     |              +--ro hello-adjacencies* [adjacent-address]
      |     |                 +--ro adjacent-address    inet:ipv4-address
      |     |                 +--ro flag*               identityref
      |     |                 +--ro hello-holdtime
      |     |                 |  +--ro adjacent?     uint16
      |     |                 |  +--ro negotiated?   uint16
      |     |                 |  +--ro remaining?    uint16
      |     |                 +--ro next-hello?         uint16
      |     |                 +--ro statistics
      |     |                 |  +--ro discontinuity-time    yang:date-and-time
      |     |                 |  +--ro hello-received?       yang:counter64
      |     |                 |  +--ro hello-dropped?        yang:counter64
      |     |                 +--ro peer?               leafref
      |     +--rw targeted
      |        +--ro state
      |        |  +--ro hello-holdtime?   uint16
      |        |  +--ro hello-interval?   uint16
      |        |  +--ro hello-accept
      |        |     +--ro enable?   boolean
      |        +--rw address-families
      |           +--rw ipv4
      |              +--ro state
      |              |  +--ro hello-adjacencies* [local-address adjacent-address
]
      |              |     +--ro local-address       inet:ipv4-address
      |              |     +--ro adjacent-address    inet:ipv4-address
      |              |     +--ro flag*               identityref
      |              |     +--ro hello-holdtime
      |              |     |  +--ro adjacent?     uint16
      |              |     |  +--ro negotiated?   uint16
      |              |     |  +--ro remaining?    uint16
      |              |     +--ro next-hello?         uint16
      |              |     +--ro statistics
      |              |     |  +--ro discontinuity-time    yang:date-and-time
      |              |     |  +--ro hello-received?       yang:counter64
      |              |     |  +--ro hello-dropped?        yang:counter64
      |              |     +--ro peer?               leafref
      |              +--rw target* [adjacent-address]
      |                 +--rw adjacent-address    inet:ipv4-address
      |                 +--ro state
      |                    +--ro enable?          boolean
      |                    +--ro local-address?   inet:ipv4-address
      +--rw peers
         +--ro state
         |  +--ro authentication
         |  |  +--ro (auth-type-selection)?
         |  |     +--:(auth-key)
         |  |        +--ro md5-key?   string
         |  +--ro capability
         |  +--ro session-ka-holdtime?   uint16
         |  +--ro session-ka-interval?   uint16
         +--rw peer* [lsr-id]
            +--rw lsr-id    yang:dotted-quad
            +--ro state
               +--ro authentication
               |  +--ro (auth-type-selection)?
               |     +--:(auth-key)
               |        +--ro md5-key?   string
               +--ro address-families
               |  +--ro ipv4
               |     +--ro hello-adjacencies* [local-address adjacent-address]
               |        +--ro local-address       inet:ipv4-address
               |        +--ro adjacent-address    inet:ipv4-address
               |        +--ro flag*               identityref
               |        +--ro hello-holdtime
               |        |  +--ro adjacent?     uint16
               |        |  +--ro negotiated?   uint16
               |        |  +--ro remaining?    uint16
               |        +--ro next-hello?         uint16
               |        +--ro statistics
               |        |  +--ro discontinuity-time    yang:date-and-time
               |        |  +--ro hello-received?       yang:counter64
               |        |  +--ro hello-dropped?        yang:counter64
               |        +--ro interface?          mpls-interface-ref
               +--ro label-advertisement-mode
               |  +--ro local?        label-adv-mode
               |  +--ro peer?         label-adv-mode
               |  +--ro negotiated?   label-adv-mode
               +--ro next-keep-alive?            uint16
               +--ro peer-ldp-id?                yang:dotted-quad
               +--ro received-peer-state
               |  +--ro graceful-restart
               |  |  +--ro enable?            boolean
               |  |  +--ro reconnect-time?    uint16
               |  |  +--ro recovery-time?     uint16
               |  +--ro capability
               |     +--ro end-of-lib
               |     |  +--ro enable?   boolean
               |     +--ro typed-wildcard-fec
               |     |  +--ro enable?   boolean
               |     +--ro upstream-label-assignment
               |        +--ro enable?   boolean
               +--ro session-holdtime
               |  +--ro peer?         uint16
               |  +--ro negotiated?   uint16
               |  +--ro remaining?    uint16
               +--ro session-state?              enumeration
               +--ro tcp-connection
               |  +--ro local-address?    inet:ip-address
               |  +--ro local-port?       inet:port-number
               |  +--ro remote-address?   inet:ip-address
               |  +--ro remote-port?      inet:port-number
               +--ro up-time?                    string
               +--ro statistics
                  +--ro discontinuity-time          yang:date-and-time
                  +--ro received
                  |  +--ro total-octets?          yang:counter64
                  |  +--ro total-messages?        yang:counter64
                  |  +--ro address?               yang:counter64
                  |  +--ro address-withdraw?      yang:counter64
                  |  +--ro initialization?        yang:counter64
                  |  +--ro keepalive?             yang:counter64
                  |  +--ro label-abort-request?   yang:counter64
                  |  +--ro label-mapping?         yang:counter64
                  |  +--ro label-release?         yang:counter64
                  |  +--ro label-request?         yang:counter64
                  |  +--ro label-withdraw?        yang:counter64
                  |  +--ro notification?          yang:counter64
                  +--ro sent
                  |  +--ro total-octets?          yang:counter64
                  |  +--ro total-messages?        yang:counter64
                  |  +--ro address?               yang:counter64
                  |  +--ro address-withdraw?      yang:counter64
                  |  +--ro initialization?        yang:counter64
                  |  +--ro keepalive?             yang:counter64
                  |  +--ro label-abort-request?   yang:counter64
                  |  +--ro label-mapping?         yang:counter64
                  |  +--ro label-release?         yang:counter64
                  |  +--ro label-request?         yang:counter64
                  |  +--ro label-withdraw?        yang:counter64
                  |  +--ro notification?          yang:counter64
                  +--ro total-addresses?            uint32
                  +--ro total-labels?               uint32
                  +--ro total-fec-label-bindings?   uint32
```

Figure 7

5.1.2. Extended

   Following is a simplified graphical representation of the extended
   data model for LDP operational state.

```
module: ietf-mpls-ldp
augment /rt:routing/rt:control-plane-protocols:
   +--rw mpls-ldp!
      +--rw global
      |  +--ro state
      |  |  +--ro capability
      |  |  |  +--ro ldp-ext:end-of-lib {capability-end-of-lib}?
      |  |  |  |  +--ro ldp-ext:enable?   boolean
      |  |  |  +--ro ldp-ext:typed-wildcard-fec {capability-typed-wildcard-fec}?
      |  |  |  |  +--ro ldp-ext:enable?   boolean
      |  |  |  +--ro ldp-ext:upstream-label-assignment {capability-upstream-labe
l-assignment}?
      |  |  |     +--ro ldp-ext:enable?   boolean
      |  |  +--ro graceful-restart
      |  |  |  +--ro ldp-ext:helper-enable?   boolean {graceful-restart-helper-m
ode}?
      |  |  +--ro ldp-ext:igp-synchronization-delay?   uint16
      |  |  +--ro ldp-ext:label-policy
      |  |     +--ro ldp-ext:advertise
      |  |        +--ro ldp-ext:egress-explicit-null
      |  |           +--ro ldp-ext:enable?   boolean
      |  +--rw address-families
      |  |  +--rw ipv4
      |  |  |  +--ro state
      |  |  |     +--ro label-policy
      |  |  |     |  +--ro advertise
      |  |  |     |  |  +--ro ldp-ext:prefix-list?    prefix-list-ref
      |  |  |     |  +--ro ldp-ext:accept
      |  |  |     |  |  +--ro ldp-ext:prefix-list?   prefix-list-ref
      |  |  |     |  +--ro ldp-ext:assign {policy-label-assignment-config}?
      |  |  |     |     +--ro ldp-ext:independent-mode
      |  |  |     |     |  +--ro ldp-ext:prefix-list?   prefix-list-ref
      |  |  |     |     +--ro ldp-ext:ordered-mode {policy-ordered-label-config}
?
      |  |  |     |        +--ro ldp-ext:egress-prefix-list?   prefix-list-ref
      |  |  |     +--ro ldp-ext:transport-address?        inet:ipv4-address
      |  |  +--rw ldp-ext:ipv6
      |  |     +--ro ldp-ext:state
      |  |        +--ro ldp-ext:enable?                          boolean
      |  |        +--ro ldp-ext:label-distribution-controlmode?   enumeration
      |  |        +--ro ldp-ext:label-policy
      |  |        |  +--ro ldp-ext:advertise
      |  |        |  |  +--ro ldp-ext:egress-explicit-null
      |  |        |  |  |  +--ro ldp-ext:enable?   boolean
      |  |        |  |  +--ro ldp-ext:prefix-list?            prefix-list-ref
      |  |        |  +--ro ldp-ext:accept
      |  |        |  |  +--ro ldp-ext:prefix-list?   prefix-list-ref
      |  |        |  +--ro ldp-ext:assign {policy-label-assignment-config}?
      |  |        |     +--ro ldp-ext:independent-mode
      |  |        |     |  +--ro ldp-ext:prefix-list?   prefix-list-ref
      |  |        |     +--ro ldp-ext:ordered-mode {policy-ordered-label-config}
?
      |  |        |        +--ro ldp-ext:egress-prefix-list?   prefix-list-ref
      |  |        +--ro ldp-ext:bindings
      |  |        |  +--ro ldp-ext:address* [address]
      |  |        |  |  +--ro ldp-ext:address               inet:ipv6-address
      |  |        |  |  +--ro ldp-ext:advertisement-type?   advertised-received
      |  |        |  |  +--ro ldp-ext:peer?                 leafref
      |  |        |  +--ro ldp-ext:fec-label* [fec]
      |  |        |     +--ro ldp-ext:fec     inet:ipv6-prefix
      |  |        |     +--ro ldp-ext:peer* [peer advertisement-type]
      |  |        |        +--ro ldp-ext:peer                  leafref
      |  |        |        +--ro ldp-ext:advertisement-type    advertised-receiv
ed
      |  |        |        +--ro ldp-ext:label?                rt-types:mpls-lab
el
      |  |        |        +--ro ldp-ext:used-in-forwarding?   boolean
      |  |        +--ro ldp-ext:transport-address?                inet:ipv6-addr
ess
      |  +--rw discovery
      |  |  +--rw interfaces
      |  |  |  +--rw interface* [interface]
      |  |  |     +--rw interface          mpls-interface-ref
      |  |  |     +--ro state
      |  |  |     |  +--ro ldp-ext:hello-holdtime?             uint16
      |  |  |     |  +--ro ldp-ext:hello-interval?             uint16
      |  |  |     |  +--ro ldp-ext:igp-synchronization-delay?   uint16 {per-inte
rface-timer-config}?
      |  |  |     +--rw address-families
      |  |  |        +--rw ipv4
      |  |  |        |  +--ro state
      |  |  |        |     +--ro ldp-ext:transport-address?   union
      |  |  |        +--rw ldp-ext:ipv6
      |  |  |           +--ro ldp-ext:state
      |  |  |              +--ro ldp-ext:enable?              boolean
      |  |  |              +--ro ldp-ext:hello-adjacencies* [adjacent-address]
      |  |  |              |  +--ro ldp-ext:adjacent-address    inet:ipv6-addres
s
      |  |  |              |  +--ro ldp-ext:flag*               identityref
      |  |  |              |  +--ro ldp-ext:hello-holdtime
      |  |  |              |  |  +--ro ldp-ext:adjacent?     uint16
      |  |  |              |  |  +--ro ldp-ext:negotiated?   uint16
      |  |  |              |  |  +--ro ldp-ext:remaining?    uint16
      |  |  |              |  +--ro ldp-ext:next-hello?         uint16
      |  |  |              |  +--ro ldp-ext:statistics
      |  |  |              |  |  +--ro ldp-ext:discontinuity-time    yang:date-a
nd-time
      |  |  |              |  |  +--ro ldp-ext:hello-received?       yang:counte
r64
      |  |  |              |  |  +--ro ldp-ext:hello-dropped?        yang:counte
r64
      |  |  |              |  +--ro ldp-ext:peer?               leafref
      |  |  |              +--ro ldp-ext:transport-address?   union
      |  |  +--rw targeted
      |  |     +--ro state
      |  |     |  +--ro hello-accept
      |  |     |     +--ro ldp-ext:neighbor-list?   neighbor-list-ref {policy-ta
rgeted-discovery-config}?
      |  |     +--rw address-families
      |  |        +--rw ldp-ext:ipv6
      |  |           +--ro ldp-ext:state
      |  |           |  +--ro ldp-ext:hello-adjacencies* [local-address adjacent
-address]
      |  |           |     +--ro ldp-ext:local-address       inet:ipv6-address
      |  |           |     +--ro ldp-ext:adjacent-address    inet:ipv6-address
      |  |           |     +--ro ldp-ext:flag*               identityref
      |  |           |     +--ro ldp-ext:hello-holdtime
      |  |           |     |  +--ro ldp-ext:adjacent?     uint16
      |  |           |     |  +--ro ldp-ext:negotiated?   uint16
      |  |           |     |  +--ro ldp-ext:remaining?    uint16
      |  |           |     +--ro ldp-ext:next-hello?         uint16
      |  |           |     +--ro ldp-ext:statistics
      |  |           |     |  +--ro ldp-ext:discontinuity-time    yang:date-and-
time
      |  |           |     |  +--ro ldp-ext:hello-received?       yang:counter64
      |  |           |     |  +--ro ldp-ext:hello-dropped?        yang:counter64
      |  |           |     +--ro ldp-ext:peer?               leafref
      |  |           +--rw ldp-ext:target* [adjacent-address]
      |  |              +--rw ldp-ext:adjacent-address    inet:ipv6-address
      |  |              +--ro ldp-ext:state
      |  |                 +--ro ldp-ext:enable?          boolean
      |  |                 +--ro ldp-ext:local-address?   inet:ipv6-address
      |  +--rw ldp-ext:forwarding-nexthop {forwarding-nexthop-config}?
      |     +--rw ldp-ext:interfaces
      |        +--rw ldp-ext:interface* [interface]
      |           +--rw ldp-ext:interface         ldp:mpls-interface-ref
      |           +--rw ldp-ext:address-family* [afi]
      |              +--rw ldp-ext:afi       ldp:ldp-address-family
      |              +--ro ldp-ext:state
      |                 +--ro ldp-ext:ldp-disable?   boolean
      +--rw peers
         +--ro state
         |  +--ro authentication
         |  |  +--ro (auth-type-selection)?
         |  |     +--:(ldp-ext:auth-key-chain)
         |  |        +--ro ldp-ext:key-chain?   key-chain:key-chain-ref
         |  +--ro session-ka-interval?                    uint16
         |  +--ro ldp-ext:session-downstream-on-demand {session-downstream-on-de
mand-config}?
         |     +--ro ldp-ext:enable?      boolean
         |     +--ro ldp-ext:peer-list?   peer-list-ref
         +--rw peer* [lsr-id]
            +--rw lsr-id    yang:dotted-quad
            +--ro state
               +--ro authentication
               |  +--ro (auth-type-selection)?
               |     +--:(ldp-ext:auth-key-chain)
               |        +--ro ldp-ext:key-chain?   key-chain:key-chain-ref
               +--ro address-families
               |  +--ro ipv4
               |  |  +--ro ldp-ext:label-policy
               |  |     +--ro ldp-ext:advertise
               |  |     |  +--ro ldp-ext:prefix-list?   prefix-list-ref
               |  |     +--ro ldp-ext:accept
               |  |        +--ro ldp-ext:prefix-list?   prefix-list-ref
               |  +--ro ldp-ext:ipv6
               |     +--ro ldp-ext:hello-adjacencies* [local-address adjacent-ad
dress]
               |     |  +--ro ldp-ext:local-address       inet:ipv6-address
               |     |  +--ro ldp-ext:adjacent-address    inet:ipv6-address
               |     |  +--ro ldp-ext:flag*               identityref
               |     |  +--ro ldp-ext:hello-holdtime
               |     |  |  +--ro ldp-ext:adjacent?     uint16
               |     |  |  +--ro ldp-ext:negotiated?   uint16
               |     |  |  +--ro ldp-ext:remaining?    uint16
               |     |  +--ro ldp-ext:next-hello?         uint16
               |     |  +--ro ldp-ext:statistics
               |     |  |  +--ro ldp-ext:discontinuity-time    yang:date-and-tim
e
               |     |  |  +--ro ldp-ext:hello-received?       yang:counter64
               |     |  |  +--ro ldp-ext:hello-dropped?        yang:counter64
               |     |  +--ro ldp-ext:interface?          ldp:mpls-interface-ref
               |     +--ro ldp-ext:label-policy
               |        +--ro ldp-ext:advertise
               |        |  +--ro ldp-ext:prefix-list?   prefix-list-ref
               |        +--ro ldp-ext:accept
               |           +--ro ldp-ext:prefix-list?   prefix-list-ref
               +--ro ldp-ext:admin-down?           boolean
               +--ro ldp-ext:label-policy
               |  +--ro ldp-ext:advertise
               |  |  +--ro ldp-ext:prefix-list?   prefix-list-ref
               |  +--ro ldp-ext:accept
               |     +--ro ldp-ext:prefix-list?   prefix-list-ref
               +--ro ldp-ext:graceful-restart
               |  +--ro ldp-ext:enable?           boolean
               |  +--ro ldp-ext:reconnect-time?   uint16
               |  +--ro ldp-ext:recovery-time?    uint16
               +--ro ldp-ext:session-ka-holdtime?   uint16
               +--ro ldp-ext:session-ka-interval?   uint16
```

Figure 8

5.2. Derived States

Following are main areas for which LDP operational "derived" state is defined:

Neighbor Adjacencies

Peer

Bindings (FEC-label and address)

Capabilities

5.2.1. Adjacency state

Neighbor adjacencies are per address-family hello adjacencies that are formed with neighbors as result of LDP basic or extended discovery. In terms of organization, there is a source of discovery (e.g. interface or target address) along with its associated parameters and one or more discovered neighbors along with neighbor discovery related parameters. For the basic discovery, there could be more than one discovered neighbor for a given source (interface), whereas there is at most one discovered neighbor for an extended discovery source (local-address and target-address). This is also to be noted that the reason for a targeted neighbor adjacency could be either an active source (locally configured targeted) or passive source (to allow any incoming extended/targeted hellos). A neighbor/ adjacency record also contains session-state that helps highlight whether a given adjacency has progressed to subsequent session level or to eventual peer level.

Following captures high level tree hierarchy for neighbor adjacency state.

```
   +--rw mpls-ldp!
      +--rw discovery
         +--rw interfaces
         |  +--rw interface* [interface]
         |     +--rw address-families
         |        +--rw ipv4 (or ipv6)
         |           +--ro state
         |              +--ro hello-adjacencies* [adjacent-address]
         |                 +--ro adjacent-address
         |                    . . . .
         |                    . . . .
         +--rw targeted
            +--rw address-families
                   +--rw ipv4 (or ipv6)
                      +--ro state
                         +--ro hello-adjacencies* [local-address adjacent-addres
s]
                            +--ro local-address
                            +--ro adjacent-address
                               . . . .
                               . . . .
```

Figure 9

5.2.2. Peer state

Peer related derived state is presented under peers tree. This is one of the core state that provides info on the session related parameters (mode, authentication, KA timeout etc.), TCP connection info, hello adjacencies for the peer, statistics related to messages and bindings, and capabilities exchange info.

Following captures high level tree hierarchy for peer state.

```
  +--rw mpls-ldp!
      +--rw peers
         +--rw peer* [lsr-id]
            +--rw lsr-id
            +--ro state
               +--ro session-ka-holdtime?
               +-- . . . .
               +-- . . . .
               +--ro capability
               +  +-- ro . . .
               +--ro address-families
               |  +--ro ipv4 (or ipv6)
               |     +--ro hello-adjacencies* [local-address adjacent-address]
               |        . . . .
               |        . . . .
               +--ro received-peer-state
               |  +--ro . . . .
               |  +--ro capability
               |     +--ro . . . .
               +--ro statistics
                  +-- . . . .
                  +-- received
                  |   +-- ...
                  +-- sent
                      +-- ...
```

Figure 10

5.2.3. Bindings state

Binding state provides information on LDP FEC-label bindings as well as address binding for both inbound (received) as well as outbound (advertised) direction. FEC-label bindings are presented as a FEC-centric view, and address bindings are presented as an address-centric view:

```
        FEC-Label bindings:
            FEC 200.1.1.1/32:
              advertised: local-label 16000
                peer 192.168.0.2:0
                peer 192.168.0.3:0
                peer 192.168.0.4:0
              received:
                peer 192.168.0.2:0, label 16002, used-in-forwarding=Yes
                peer 192.168.0.3:0, label 17002, used-in-forwarding=No
            FEC 200.1.1.2/32:
               . . . .
            FEC 201.1.0.0/16:
               . . . .


        Address bindings:
            Addr 1.1.1.1:
              advertised
            Addr 1.1.1.2:
              advertised
            Addr 2.2.2.2:
              received, peer 192.168.0.2
            Addr 2.2.2.22:
              received, peer 192.168.0.2
            Addr 3.3.3.3:
              received, peer 192.168.0.3
            Addr 3.3.3.33:
              received, peer 192.168.0.3
```

Figure 11

Note that all local addresses are advertised to all peers and hence no need to provide per-peer information for local address advertisement. Furthermore, note that it is easy to derive a peer-centric view for the bindings from the information already provided in this model.

Following captures high level tree hierarchy for bindings state.

```
   +--rw mpls-ldp!
      +--rw global
         +--rw address-families
            +--rw ipv4 (or ipv6)
               +--ro state
                  +--ro bindings
                     +--ro address* [address]
                     |  +--ro address
                     |  +--ro dvertisement-type?   advertised-received
                     |  +--ro peer?        leafref
                     +--ro fec-label* [fec]
                        +--ro fec     inet:ipv4-prefix
                        +--ro peer* [peer advertisement-type]
                           +--ro peer                  leafref
                           +--ro advertisement-type?   advertised-received
                           +--ro label?                rt-types:mpls-label
                           +--ro used-in-forwarding?   boolean
```

Figure 12

5.2.4. Capabilities state

LDP capabilities state comprise two types of information - global information (such as timer etc.), and per-peer information.

Following captures high level tree hierarchy for LDP capabilities state.

```
      +--rw mpls-ldp!
         +--rw global
         |  +--ro state
         |     +--ro capability
         |        +--ro . . . .
         |        +--ro . . . .
         +--rw peers
            +--rw peer* [lsr-id]
               +--rw lsr-id    yang:dotted-quad
               +--ro state
                  +--ro received-peer-state
                     +--ro capability
                        +--ro . . . .
                        +--ro . . . .
```

Figure 13

6. Notifications

This model defines a list of notifications to inform client of important events detected during the protocol operation. These events include events related to changes in the operational state of an LDP peer, hello adjacency, and FEC etc. It is to be noted that an LDP FEC is treated as operational (up) as long as it has at least 1 NHLFE with outgoing label.

Following is a simplified graphical representation of the data model for LDP notifications.

```
   module: ietf-mpls-ldp
   notifications:
      +---n mpls-ldp-peer-event
      |  +--ro event-type?   oper-status-event-type
      |  +--ro peer-ref?     leafref
      +---n mpls-ldp-hello-adjacency-event
      |  +--ro event-type?   oper-status-event-type
      |  +--ro (hello-adjacency-type)?
      |     +--:(targeted)
      |     |  +--ro targeted
      |     |     +--ro target-address?   inet:ip-address
      |     +--:(link)
      |        +--ro link
      |           +--ro next-hop-interface?   mpls-interface-ref
      |           +--ro next-hop-address?     inet:ip-address
      +---n mpls-ldp-fec-event
         +--ro event-type?   oper-status-event-type
         +--ro prefix?       inet:ip-prefix
```

Figure 14

7. Actions

This model defines a list of rpcs that allow performing an action or executing a command on the protocol. For example, it allows to clear (reset) LDP peers, hello-adjacencies, and statistics. The model makes an effort to provide different level of control so that a user is able to either clear all, or clear all for a given type, or clear a specific entity.

Following is a simplified graphical representation of the data model for LDP actions.

```
   module: ietf-mpls-ldp
   rpcs:
      +---x mpls-ldp-clear-peer
      |  +---w input
      |     +---w lsr-id?   union
      +---x mpls-ldp-clear-hello-adjacency
      |  +---w input
      |     +---w hello-adjacency
      |        +---w (hello-adjacency-type)?
      |           +--:(targeted)
      |           |  +---w targeted!
      |           |     +---w target-address?   inet:ip-address
      |           +--:(link)
      |              +---w link!
      |                 +---w next-hop-interface?   mpls-interface-ref
      |                 +---w next-hop-address?     inet:ip-address
      +---x mpls-ldp-clear-peer-statistics
         +---w input
            +---w lsr-id?   union
```

Figure 15

8. Open Items

Following is a list of open items that are to be discussed and addressed in future revisions of this document:

- o Align operational state modeling with other routing protocols and [I-D.openconfig-netmod-opstate]

- o Specify default values for configuration parameters

- o Close on augmentation off "mpls" list in "ietf-mpls" defined in [I-D.ietf-mpls-base-yang]

- o The use of grouping (templates) for bundling and grouping the configuration items is not employed in current revision, and is a subject for consideration in future.

- o Decide on which label-policy mode (global, per-af, per-peer, per-peer-per-af) to use as base.

9. YANG Specification

Following are the actual YANG definition (module) for LDP constructs defined earlier in the document.

9.1. Base

```
  <CODE BEGINS> file "ietf-mpls-ldp@2017-03-12.yang"

  module ietf-mpls-ldp {
    namespace "urn:ietf:params:xml:ns:yang:ietf-mpls-ldp";
    prefix "ldp";

    import ietf-inet-types {
      prefix "inet";
    }

    import ietf-yang-types {
      prefix "yang";
    }

    import ietf-routing {
      prefix "rt";
    }

    import ietf-routing-types {
       prefix "rt-types";
    }

    import ietf-mpls {
      prefix "mpls";
    }

    organization
      "IETF MPLS Working Group";
    contact
      "WG Web:   <http://tools.ietf.org/wg/teas/>
       WG List:  <mailto:teas@ietf.org>

       WG Chair: Loa Andersson
                 <mailto:loa@pi.nu>

       WG Chair: Ross Callon
                 <mailto:rcallon@juniper.net>

       WG Chair: George Swallow
                 <mailto:swallow.ietf@gmail.com>

       Editor:   Kamran Raza
                 <mailto:skraza@cisco.com>

       Editor:   Rajiv Asati
                 <mailto:rajiva@cisco.com>
       Editor:   Xufeng Liu
                 <mailto:Xufeng_Liu@jabil.com>

       Editor:   Santosh Esale
                 <mailto:sesale@juniper.net>

       Editor:   Xia Chen
                 <mailto:jescia.chenxia@huawei.com>

       Editor:   Himanshu Shah
                 <mailto:hshah@ciena.com>";

    description
      "This YANG module defines the essential components for the
       management of Multi-Protocol Label Switching (MPLS) Label
       Distribution Protocol (LDP). It is also the base model to
       be augmented for Multipoint LDP (mLDP).";

    revision 2017-03-12 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Data Model for MPLS LDP.";
    }

    /*
     * Typedefs
     */
    typedef ldp-address-family {
      type identityref {
        base rt:address-family;
      }
      description
        "LDP address family type.";
    }

    typedef duration32-inf {
      type union {
        type uint32;
        type enumeration {
          enum "infinite" {
            description "The duration is infinite.";
          }
        }
      }
      units seconds;
      description
        "Duration represented as 32 bit seconds with infinite.";
    }

    typedef advertised-received {
      type enumeration {
        enum advertised {
          description "Advertised information.";
        }
        enum received {
          description "Received information.";
        }
      }
      description
        "Received or advertised.";
    }

    typedef downstream-upstream {
      type enumeration {
        enum downstream {
          description "Downstream information.";
        }
        enum upstream {
          description "Upstream information.";
        }
      }
      description
        "Received or advertised.";
    }

    typedef label-adv-mode {
      type enumeration {
        enum downstream-unsolicited {
          description "Downstream Unsolicited.";
        }
        enum downstream-on-demand {
          description "Downstream on Demand.";
        }
      }
      description
        "Label Advertisement Mode.";
    }

    typedef mpls-interface-ref {
      type leafref {
        path "/rt:routing/mpls:mpls/mpls:interface/mpls:name";
      }
      description
        "This type is used by data models that need to reference
         mpls interfaces.";
    }

    typedef oper-status-event-type {
      type enumeration {
        enum up {
          value 1;
          description
            "Operational status changed to up.";
        }
        enum down {
          value 2;
          description
            "Operational status changed to down.";
        }
      }
      description "Operational status event type for notifications.";
    }

    /*
     * Identities
     */
    identity adjacency-flag-base {
      description "Base type for adjacency flags.";
    }

    identity adjacency-flag-active {
      base "adjacency-flag-base";
      description
        "This adjacency is configured and actively created.";
    }

    identity adjacency-flag-passive {
      base "adjacency-flag-base";
      description
        "This adjacency is not configured and passively accepted.";
    }

    /*
     * Groupings
     */

    grouping adjacency-state-attributes {
      description
        "Adjacency state attributes.";

      leaf-list flag {
        type identityref {
          base "adjacency-flag-base";
        }
        description "Adjacency flags.";
      }
      container hello-holdtime {
        description "Hello holdtime state.";
        leaf adjacent {
          type uint16;
          units seconds;
          description "Peer holdtime.";
        }
        leaf negotiated {
          type uint16;
          units seconds;
          description "Negotiated holdtime.";
        }
        leaf remaining {
          type uint16;
          units seconds;
          description "Remaining holdtime.";
        }
      }

      leaf next-hello {
        type uint16;
        units seconds;
        description "Time to send the next hello message.";
      }

      container statistics {
        description
          "Statistics objects.";

        leaf discontinuity-time {
          type yang:date-and-time;
          mandatory true;
          description
            "The time on the most recent occasion at which any one or
             more of this interface's counters suffered a
             discontinuity.  If no such discontinuities have occurred
             since the last re-initialization of the local management
             subsystem, then this node contains the time the local
             management subsystem re-initialized itself.";
        }

        leaf hello-received {
          type yang:counter64;
          description
            "The number of hello messages received.";
        }
        leaf hello-dropped {
          type yang:counter64;
          description
            "The number of hello messages received.";
        }
      } // statistics
    } // adjacency-state-attributes

    grouping basic-discovery-timers {
      description
        "Basic discovery timer attributes.";
      leaf hello-holdtime {
        type uint16 {
          range 15..3600;
        }
        units seconds;
        description
          "The time interval for which a LDP link Hello adjacency
           is maintained in the absence of link Hello messages from
           the LDP neighbor";
      }
      leaf hello-interval {
        type uint16 {
          range 5..1200;
        }
        units seconds;
        description
          "The interval between consecutive LDP link Hello messages
           used in basic LDP discovery";
      }
    } // basic-discovery-timers

    grouping binding-address-state-attributes {
      description
        "Address binding attributes";
      leaf advertisement-type {
        type advertised-received;
        description
          "Received or advertised.";
      }
      leaf peer {
        type leafref {
          path "../../../../../../../ldp:peers/ldp:peer/ldp:lsr-id";
        }
        must "../advertisement-type = 'received'" {
          description
            "Applicable for received address.";
        }
        description
          "LDP peer from which this address is received.";
      } // peer
    } // binding-address-state-attributes

    grouping binding-label-state-attributes {
      description
        "Label binding attributes";
      list peer {
        key "peer advertisement-type";
        description
          "List of advertised and received peers.";
        leaf peer {
          type leafref {
            path "../../../../../../../../ldp:peers/ldp:peer/"
              + "ldp:lsr-id";
          }
          description
            "LDP peer from which this binding is received,
             or to which this binding is advertised.";
        }
        leaf advertisement-type {
          type advertised-received;
          description
            "Received or advertised.";
        }
        leaf label {
          type rt-types:mpls-label;
          description
            "Advertised (outbound) or received (inbound)
             label.";
        }
        leaf used-in-forwarding {
          type boolean;
          description
            "'true' if the lable is used in forwarding.";
        }
      } // peer
    } // binding-label-state-attributes

    grouping extended-discovery-policy-attributes {
      description
        "LDP policy to control the acceptance of extended neighbor
         discovery hello messages.";
      container hello-accept {
        description
          "Extended discovery acceptance policies.";
        leaf enable {
          type boolean;
          description
            "'true' to accept; 'false' to deny.";
        }
      } // hello-accept
    } // extended-discovery-policy-attributes

    grouping extended-discovery-timers {
      description
        "Extended discovery timer attributes.";
      leaf hello-holdtime {
        type uint16 {
          range 15..3600;
        }
        units seconds;
        description
          "The time interval for which LDP targeted Hello adjacency

           is maintained in the absence of targeted Hello messages
           from an LDP neighbor.";
      }
      leaf hello-interval {
        type uint16 {
          range 5..3600;
        }
        units seconds;
        description
          "The interval between consecutive LDP targeted Hello
           messages used in extended LDP discovery.";
      }
    } // extended-discovery-timers

    grouping global-attributes {
      description "Configuration attributes at global level.";

      uses instance-attributes;
    } // global-attributes

    grouping graceful-restart-attributes {
      description
        "Graceful restart configuration attributes.";
      container graceful-restart {
        description
          "Attributes for graceful restart.";
        leaf enable {
          type boolean;
          description
```

```
            "Enable or disable graceful restart.";
        }
        leaf reconnect-time {
          type uint16 {
            range 10..1800;
          }
          units seconds;
          description
            "Specifies the time interval that the remote LDP peer
             must wait for the local LDP peer to reconnect after the
             remote peer detects the LDP communication failure.";
        }
        leaf recovery-time {
          type uint16 {
            range 30..3600;
          }
          units seconds;
          description
            "Specifies the time interval, in seconds, that the remote
             LDP peer preserves its MPLS forwarding state after
             receiving the Initialization message from the restarted
             local LDP peer.";
        }
        leaf forwarding-holdtime {
          type uint16 {
            range 30..3600;
          }
          units seconds;
          description
            "Specifies the time interval, in seconds, before the
             termination of the recovery phase.";
        }
      } // graceful-restart
    } // graceful-restart-attributes

    grouping graceful-restart-attributes-per-peer {
      description
        "Per peer graceful restart configuration attributes.";
      container graceful-restart {
        description
          "Attributes for graceful restart.";
        leaf enable {
          type boolean;
          description
            "Enable or disable graceful restart.";
        }
        leaf reconnect-time {
          type uint16 {
            range 10..1800;
          }
          units seconds;
          description
            "Specifies the time interval that the remote LDP peer
             must wait for the local LDP peer to reconnect after the
             remote peer detects the LDP communication failure.";
        }
        leaf recovery-time {
          type uint16 {
            range 30..3600;
          }
          units seconds;
          description
            "Specifies the time interval, in seconds, that the remote
             LDP peer preserves its MPLS forwarding state after
             receiving the Initialization message from the restarted
             local LDP peer.";
        }
      } // graceful-restart
    } // graceful-restart-attributes-per-peer

    grouping instance-attributes {
      description "Configuration attributes at instance level.";

      container capability {
        description "Configure capability.";
      } // capability

      uses graceful-restart-attributes;

      leaf lsr-id {
        type yang:dotted-quad;
        description "Router ID.";
      }
    } // instance-attributes

    grouping ldp-adjacency-ref {
      description
        "An absolute reference to an LDP adjacency.";
      choice hello-adjacency-type {
        description
          "Interface or targeted adjacency.";
        case targeted {
          container targeted {
            description "Targeted adjacency.";
            leaf target-address {
              type inet:ip-address;
              description
                "The target address.";
            }
          } // targeted
        }
        case link {
          container link {
            description "Link adjacency.";
            leaf next-hop-interface {
              type mpls-interface-ref;
              description
                "Interface connecting to next-hop.";
            }
            leaf next-hop-address {
              type inet:ip-address;
              must "../next-hop-interface" {
                description
                  "Applicable when interface is specified.";

              }
              description
                "IP address of next-hop.";
            }
          } // link
        }
      }
    } // ldp-adjacency-ref

    grouping ldp-fec-event {
      description
        "A LDP FEC event.";
      leaf prefix {
        type inet:ip-prefix;
        description
          "FEC.";
      }
    } // ldp-fec-event

    grouping ldp-peer-ref {
      description
        "An absolute reference to an LDP peer.";
      leaf peer-ref {
        type leafref {
          path "/rt:routing/rt:control-plane-protocols/mpls-ldp/"
            + "peers/peer/lsr-id";
        }
        description
          "Reference to an LDP peer.";
      }
    } // ldp-peer-ref

    grouping peer-attributes {
      description "Peer configuration attributes.";

      leaf session-ka-holdtime {
        type uint16 {
          range 45..3600;
        }
        units seconds;
        description
          "The time interval after which an inactive LDP session
           terminates and the corresponding TCP session closes.
           Inactivity is defined as not receiving LDP packets from the
           peer.";
      }
      leaf session-ka-interval {
        type uint16 {
          range 15..1200;
        }
        units seconds;
        description
          "The interval between successive transmissions of keepalive
           packets. Keepalive packets are only sent in the absence of
           other LDP packets transmitted over the LDP session.";
      }
    } // peer-attributes

    grouping peer-authentication {
      description
        "Peer authentication container.";
  /*
      leaf session-authentication-md5-password {
        type string {
          length "1..80";
        }
        description
          "Assigns an encrypted MD5 password to an LDP
           peer";
      } // md5-password
  */
      container authentication {
        description "Containing authentication information.";
        choice auth-type-selection {
          description
            "Options for expressing authentication setting.";
          case auth-key {
            leaf md5-key {
              type string;
              description
                "MD5 Key string.";
            }
          }
        }
      } // authentication
    } // peer-authentication

    grouping peer-state-derived {
      description "Peer derived state attributes.";

      container label-advertisement-mode {
        description "Label advertisement mode state.";
        leaf local {
          type label-adv-mode;
          description
            "Local Label Advertisement Mode.";
        }
        leaf peer {
          type label-adv-mode;
          description
            "Peer Label Advertisement Mode.";
        }
        leaf negotiated {
          type label-adv-mode;
          description
            "Negotiated Label Advertisement Mode.";
        }
      }
      leaf next-keep-alive {
        type uint16;
        units seconds;
        description "Time to send the next KeepAlive message.";
      }

      leaf peer-ldp-id {
        type yang:dotted-quad;
        description "Peer LDP ID.";
      }

      container received-peer-state {
        description "Peer features.";

        uses graceful-restart-attributes-per-peer;

        container capability {
          description "Configure capability.";
          container end-of-lib {
            description
              "Configure end-of-lib capability.";
            leaf enable {
              type boolean;
              description
                "Enable end-of-lib capability.";
            }
          }
          container typed-wildcard-fec {
            description
              "Configure typed-wildcard-fec capability.";
            leaf enable {
              type boolean;
              description
                "Enable typed-wildcard-fec capability.";
            }
          }
          container upstream-label-assignment {
            description
              "Configure upstream label assignment capability.";
            leaf enable {
              type boolean;
              description
                "Enable upstream label assignment.";
            }
          }
        } // capability
      } // received-peer-state

      container session-holdtime {
        description "Session holdtime state.";
        leaf peer {
          type uint16;
          units seconds;
          description "Peer holdtime.";
        }
        leaf negotiated {
          type uint16;
          units seconds;
          description "Negotiated holdtime.";
        }
        leaf remaining {
          type uint16;
          units seconds;
          description "Remaining holdtime.";
        }
      } // session-holdtime

      leaf session-state {
        type enumeration {
          enum non-existent {
            description "NON EXISTENT state. Transport disconnected.";
          }
          enum initialized {
            description "INITIALIZED state.";
          }
          enum openrec {
            description "OPENREC state.";
          }
          enum opensent {
            description "OPENSENT state.";
          }
          enum operational {
            description "OPERATIONAL state.";
          }
        }
        description
          "Representing the operational status.";
      }

      container tcp-connection {
        description "TCP connection state.";
        leaf local-address {
          type inet:ip-address;
          description "Local address.";
        }
        leaf local-port {
          type inet:port-number;
          description "Local port.";
        }
        leaf remote-address {
          type inet:ip-address;
          description "Remote address.";
        }
        leaf remote-port {
          type inet:port-number;
          description "Remote port.";
        }
      } // tcp-connection

      leaf up-time {
        type string;
        description "Up time. The interval format in ISO 8601.";
      }
      container statistics {
        description
          "Statistics objects.";

        leaf discontinuity-time {
          type yang:date-and-time;
          mandatory true;
          description
            "The time on the most recent occasion at which any one or
             more of this interface's counters suffered a
             discontinuity.  If no such discontinuities have occurred
             since the last re-initialization of the local management
             subsystem, then this node contains the time the local
             management subsystem re-initialized itself.";
        }

        container received {
          description "Inbound statistics.";
          uses statistics-peer-received-sent;
        }
        container sent {
          description "Outbound statistics.";
          uses statistics-peer-received-sent;
        }

        leaf total-addresses {
          type uint32;
          description
            "The number of learned addresses.";
        }
        leaf total-labels {
          type uint32;
          description
            "The number of learned labels.";
        }
        leaf total-fec-label-bindings {
          type uint32;
          description
            "The number of learned label-address bindings.";
        }
      } // statistics
    } // peer-state-derived

    grouping policy-container {
      description
        "LDP policy attributes.";
      container label-policy {
        description
```

```
          "Label policy attributes.";
        container advertise {
          description
            "Label advertising policies.";
          container egress-explicit-null {
            description
              "Enables an egress router to advertise an
               explicit null label (value 0) in place of an
               implicit null label (value 3) to the
               penultimate hop router.";
            leaf enable {
              type boolean;
              description
                "'true' to enable explicit null.";
            }
          }
        } // advertise
      } // label-policy
    } // policy-container

    grouping statistics-peer-received-sent {
      description
        "Inbound and outbound statistic counters.";
      leaf total-octets {
        type yang:counter64;
        description
          "The total number of octets sent or received.";
      }
      leaf total-messages {
        type yang:counter64;
        description
          "The number of messages sent or received.";
      }
      leaf address {
        type yang:counter64;
        description
          "The number of address messages sent or received.";
      }
      leaf address-withdraw {
        type yang:counter64;
        description
          "The number of address-withdraw messages sent or received.";
      }
      leaf initialization {
        type yang:counter64;
        description
          "The number of initialization messages sent or received.";
      }
      leaf keepalive {
        type yang:counter64;
        description
          "The number of keepalive messages sent or received.";
      }
      leaf label-abort-request {
        type yang:counter64;
        description
          "The number of label-abort-request messages sent or
           received.";
      }
      leaf label-mapping {
        type yang:counter64;
        description
          "The number of label-mapping messages sent or received.";
      }
      leaf label-release {
        type yang:counter64;
        description
          "The number of label-release messages sent or received.";
      }
      leaf label-request {
        type yang:counter64;
        description
          "The number of label-request messages sent or received.";
      }
      leaf label-withdraw {
        type yang:counter64;
        description
          "The number of label-withdraw messages sent or received.";
      }
      leaf notification {
        type yang:counter64;
        description
          "The number of messages sent or received.";
      }
    } // statistics-peer-received-sent

    /*
     * Configuration data nodes
     */

    augment "/rt:routing/rt:control-plane-protocols" {
      description "LDP augmentation.";

      container mpls-ldp {
        presence "Container for LDP protocol.";
        description
```

```
          "Container for LDP protocol.";

        container global {
          description
            "Global attributes for LDP.";
          container config {
            description
              "Configuration data.";
            uses global-attributes;
          }
          container state {
            config false;
            description
              "Operational state data.";
            uses global-attributes;
          }

          container address-families {
            description
              "Container for address families.";
            container ipv4 {
              presence
                "Present if IPv4 is enabled, unless the 'enable'
                 leaf is set to 'false'";
              description
                "IPv4 address family.";
              container config {
                description
                  "Configuration data.";
                leaf enable {
                  type boolean;
                  default true;
                  description
                    "'true' to enable the address family.";
                }
                uses policy-container;
              }
              container state {
                config false;
                description
                  "Operational state data.";
                leaf enable {
                  type boolean;
                  description
                    "'true' to enable the address family.";
                }
                leaf label-distribution-controlmode {
                  type enumeration {
                    enum independent {
                      description
                      "Independent label distribution control.";
                    }
                    enum Ordered {
                      description
                      "Ordered Label Distribution Control.";
                    }
                  }
                  description
                    "Label distribution control mode.";
                  reference
                    "RFC5036: LDP Specification. Sec 2.6.";
                }

                uses policy-container;

                // ipv4 bindings
                container bindings {
                  description
                    "LDP address and label binding information.";
                  list address {
                    key "address";
                    description
                      "List of address bindings.";
                    leaf address {
                      type inet:ipv4-address;
                      description
                        "Binding address.";
                    }
                    uses binding-address-state-attributes;
                  } // binding-address

                  list fec-label {
                    key "fec";
                    description
                      "List of label bindings.";
                    leaf fec {
                      type inet:ipv4-prefix;
                      description
                        "Prefix FEC.";
                    }
                    uses binding-label-state-attributes;
                  } // fec-label
                } // bindings
              } // state
            } // ipv4
          } // address-families
          container discovery {
            description
              "Neibgbor discovery configuration.";

            container interfaces {
              description
                "A list of interfaces for basic descovery.";
              container config {
                description
                  "Configuration data.";
                uses basic-discovery-timers;
              }
              container state {
                config false;
                description

                  "Operational state data.";
                uses basic-discovery-timers;
              }

              list interface {
                key "interface";
                description
                  "List of LDP interfaces.";
                leaf interface {
                  type mpls-interface-ref;
                  description
                    "Interface.";
                }
                container state {
                  config false;
                  description
                    "Operational state data.";
                  leaf next-hello {
                    type uint16;
                    units seconds;
                    description "Time to send the next hello message.";
                  }
                } // state

                container address-families {
                  description
                    "Container for address families.";
                  container ipv4 {
                    presence
                      "Present if IPv4 is enabled, unless the 'enable'
                       leaf is set to 'false'";
                    description
                      "IPv4 address family.";
                    container config {
                      description
                        "Configuration data.";
                      leaf enable {
                        type boolean;
                        default true;
                        description
                          "Enable the address family on the interface.";
                      }
                    }

                    container state {
                      config false;
                      description
                        "Operational state data.";
                      leaf enable {
                        type boolean;
                        description
                          "Enable the address family on the interface.";
                      }

                      // ipv4
                      list hello-adjacencies {
                        key "adjacent-address";
                        description "List of hello adjacencies.";

                        leaf adjacent-address {
                          type inet:ipv4-address;
                          description
                            "Neighbor address of the hello adjacency.";
                        }

                        uses adjacency-state-attributes;

                        leaf peer {
                          type leafref {
                            path "../../../../../../../../../../"
                              + "peers/peer/lsr-id";
                          }
                          description
                            "LDP peer from this adjacency.";
                        }
                      } // hello-adjacencies
                    } // state
                  } // ipv4
                } // address-families
              } // list interface
            } // interfaces

            container targeted
            {
              description
                "A list of targeted neighbors for extended discovery.";
              container config {

                description
                  "Configuration data.";
                uses extended-discovery-timers;
                uses extended-discovery-policy-attributes;
              }
              container state {
                config false;
                description
                  "Operational state data.";
                uses extended-discovery-timers;
                uses extended-discovery-policy-attributes;
              }

              container address-families {
                description
                  "Container for address families.";
                container ipv4 {
                  presence
                    "Present if IPv4 is enabled.";
                  description
                    "IPv4 address family.";
                  container state {
                    config false;
                    description
                      "Operational state data.";

                    list hello-adjacencies {
                      key "local-address adjacent-address";
                      description "List of hello adjacencies.";

                      leaf local-address {
                        type inet:ipv4-address;
                        description
                          "Local address of the hello adjacency.";
                      }
                      leaf adjacent-address {
                        type inet:ipv4-address;
                        description
                          "Neighbor address of the hello adjacency.";
                      }
                      uses adjacency-state-attributes;

                      leaf peer {
                        type leafref {
                          path "../../../../../../../../peers/peer/"
                            + "lsr-id";
                        }
                        description
                          "LDP peer from this adjacency.";
                      }
                    } // hello-adjacencies
                  } // state

                  list target {
                    key "adjacent-address";
                    description
                      "Targeted discovery params.";

                    leaf adjacent-address {
                      type inet:ipv4-address;
                      description
                        "Configures a remote LDP neighbor and enables
                         extended LDP discovery of the specified
                         neighbor.";
                    }
                    container config {
                      description
                        "Configuration data.";
                      leaf enable {
                        type boolean;
                        description
                          "Enable the target.";
                      }
                      leaf local-address {
                        type inet:ipv4-address;
                        description
                          "The local address.";
                      }
                    }
                    container state {
                      config false;
                      description
                        "Operational state data.";
                      leaf enable {
                        type boolean;
                        description
                          "Enable the target.";
                      }
```

```
                      leaf local-address {
                        type inet:ipv4-address;
                        description
                          "The local address.";
                      }
                    } // state
                  } // target
                } // ipv4
              } // address-families
            } // targeted
          } // discovery
        } // global

        container peers {
          description
            "Peers configuration attributes.";

          container config {
            description
              "Configuration data.";
            uses peer-authentication;
            uses peer-attributes;
          }
          container state {
            config false;
            description
              "Operational state data.";
            uses peer-authentication;
            uses peer-attributes;
          }

          list peer {
            key "lsr-id";
            description
              "List of peers.";

            leaf lsr-id {
              type yang:dotted-quad;
              description "LSR ID.";
            }

            container config {
              description
                "Configuration data.";

              uses peer-authentication;
              container capability {
                description
                  "Per peer capability";
              }
            }
            container state {
              config false;
              description
                "Operational state data.";

              uses peer-authentication;
              container capability {
                description
                  "Per peer capability";
              }

              container address-families {
                description
                  "Per-vrf per-af params.";
                container ipv4 {
                  presence
                    "Present if IPv4 is enabled.";
                  description
                    "IPv4 address family.";

                  list hello-adjacencies {
                    key "local-address adjacent-address";
                    description "List of hello adjacencies.";

                    leaf local-address {
                      type inet:ipv4-address;
                      description
                        "Local address of the hello adjacency.";
                    }
                    leaf adjacent-address {
                      type inet:ipv4-address;
                      description
                        "Neighbor address of the hello adjacency.";
                    }

                    uses adjacency-state-attributes;

                    leaf interface {
                      type mpls-interface-ref;
                      description "Interface for this adjacency.";
                    }
                  } // hello-adjacencies
                } // ipv4
              } // address-families
```

```
              uses peer-state-derived;
            } // state
          } // list peer
        } // peers
      } // container mpls-ldp
    }

    /*
     * RPCs
     */
    rpc mpls-ldp-clear-peer {
      description
        "Clears the session to the peer.";
      input {
        leaf lsr-id {
          type union {
            type yang:dotted-quad;
            type uint32;
          }
          description
            "LSR ID of peer to be cleared. If this is not provided
             then all peers are cleared";
        }
      }
    }

    rpc mpls-ldp-clear-hello-adjacency {
      description
        "Clears the hello adjacency";
      input {
        container hello-adjacency {
          description
            "Link adjacency or targettted adjacency. If this is not
             provided then all hello adjacencies are cleared";
          choice hello-adjacency-type {
            description "Adjacency type.";
            case targeted {
              container targeted {
                presence "Present to clear targeted adjacencies.";
                description
                  "Clear targeted adjacencies.";
                leaf target-address {
                  type inet:ip-address;
                  description
                    "The target address. If this is not provided then
                     all targeted adjacencies are cleared";
                }
              } // targeted
            }
            case link {
              container link {
                presence "Present to clear link adjacencies.";
                description
                  "Clear link adjacencies.";
                leaf next-hop-interface {
                  type mpls-interface-ref;
                  description

                    "Interface connecting to next-hop. If this is not
                     provided then all link adjacencies are cleared.";
                }
                leaf next-hop-address {
                  type inet:ip-address;
                  must "../next-hop-interface" {
                    description
                      "Applicable when interface is specified.";
                  }
                  description
                    "IP address of next-hop. If this is not provided
                     then adjacencies to all next-hops on the given
                     interface are cleared.";
                } // next-hop-address
              } // link
            }
          }
        }
      }
    }

    rpc mpls-ldp-clear-peer-statistics {
      description
        "Clears protocol statistics (e.g. sent and received
         counters).";
      input {
        leaf lsr-id {
          type union {
            type yang:dotted-quad;
            type uint32;
          }
          description
            "LSR ID of peer whose statistic are to be cleared.
             If this is not provided then all peers statistics are
             cleared";
        }
      }
    }
    /*
     * Notifications
     */
    notification mpls-ldp-peer-event {

      description
        "Notification event for a change of LDP peer operational
         status.";
      leaf event-type {
        type oper-status-event-type;
        description "Event type.";
      }
      uses ldp-peer-ref;
    }

    notification mpls-ldp-hello-adjacency-event {
      description
        "Notification event for a change of LDP adjacency operational
         status.";
      leaf event-type {
        type oper-status-event-type;
        description "Event type.";
      }
      uses ldp-adjacency-ref;
    }

    notification mpls-ldp-fec-event {
      description
        "Notification event for a change of FEC status.";
      leaf event-type {
        type oper-status-event-type;
        description "Event type.";
      }
      uses ldp-fec-event;
    }
  }

  <CODE ENDS>
```

Figure 16

9.2. Extended

```
   <CODE BEGINS> file "ietf-mpls-ldp-extended@2017-03-12.yang"
   module ietf-mpls-ldp-extended {
     namespace "urn:ietf:params:xml:ns:yang:ietf-mpls-ldp-extended";
     prefix "ldp-ext";

     import ietf-inet-types {
       prefix "inet";
     }
     import ietf-routing {
       prefix "rt";
     }

     import ietf-routing-types {
       prefix "rt-types";
     }

     import ietf-key-chain {
       prefix "key-chain";
     }

     import ietf-mpls-ldp {
       prefix "ldp";
     }

     organization
       "IETF MPLS Working Group";
     contact
       "WG Web:   <http://tools.ietf.org/wg/teas/>
        WG List:  <mailto:teas@ietf.org>

        WG Chair: Loa Andersson
                  <mailto:loa@pi.nu>

        WG Chair: Ross Callon
                  <mailto:rcallon@juniper.net>

        WG Chair: George Swallow
                  <mailto:swallow.ietf@gmail.com>

        Editor:   Kamran Raza
                  <mailto:skraza@cisco.com>

        Editor:   Rajiv Asati
                  <mailto:rajiva@cisco.com>

        Editor:   Xufeng Liu
                  <mailto:Xufeng_Liu@jabil.com>

        Editor:   Santosh Esale
                  <mailto:sesale@juniper.net>

        Editor:   Xia Chen
```

```
                  <mailto:jescia.chenxia@huawei.com>

        Editor:   Himanshu Shah
                  <mailto:hshah@ciena.com>";

     description
       "This YANG module defines the essential components for the
        management of Multi-Protocol Label Switching (MPLS) Label
        Distribution Protocol (LDP). It is also the base model to
        be augmented for Multipoint LDP (mLDP).";

     revision 2017-03-12 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: YANG Data Model for MPLS LDP.";
     }

     /*
      * Features
      */
     feature all-af-policy-config {
       description
         "This feature indicates that the system allows to configure
          policies that are applied to all address families.";
     }

     feature capability-end-of-lib {
       description
         "This feature indicates that the system allows to configure
          LDP end-of-lib capability.";
     }

     feature capability-typed-wildcard-fec {
       description
         "This feature indicates that the system allows to configure
          LDP typed-wildcard-fec capability.";
     }

     feature capability-upstream-label-assignment {
       description
         "This feature indicates that the system allows to configure
          LDP upstream label assignment capability.";
     }

     feature forwarding-nexthop-config {
       description
         "This feature indicates that the system allows to configure
```

```
          forwarding nexthop on interfaces.";
     }

     feature graceful-restart-helper-mode {
       description
         "This feature indicates that the system supports graceful
          restart helper mode.";
     }

     feature per-interface-timer-config {
       description
         "This feature indicates that the system allows to configure
          interface hello timers at the per-interface level.";
     }

     feature per-peer-graceful-restart-config {
       description
         "This feature indicates that the system allows to configure
          graceful restart at the per-peer level.";
     }

     feature per-peer-session-attributes-config {
       description
         "This feature indicates that the system allows to configure
          session attributes at the per-peer level.";
     }

     feature policy-label-assignment-config {
       description
         "This feature indicates that the system allows to configure
          policies to assign labels according to certain prefixes.";
     }

     feature policy-ordered-label-config {
       description
         "This feature indicates that the system allows to configure
          ordered label policies.";
     }

     feature policy-targeted-discovery-config {
       description
         "This feature indicates that the system allows to configure
          policies to control the acceptance of targeted neighbor
          discovery hello messages.";
     }

     feature session-downstream-on-demand-config {
       description
```

```
         "This feature indicates that the system allows to configure
          session downstream-on-demand";
     }

     /*
      * Typedefs
      */
     typedef neighbor-list-ref {
       type string;
       description
         "A type for a reference to a neighbor list.";
     }

     typedef prefix-list-ref {
       type string;
       description
         "A type for a reference to a prefix list.";
     }

     typedef peer-list-ref {
       type string;
       description
         "A type for a reference to a peer list.";
     }

     /*
      * Identities
      */

     /*
      * Groupings
      */
     grouping address-family-ipv4-augment {
       description "Augmentation to address family IPv4.";

       leaf transport-address {
         type inet:ipv4-address;
         description
           "The transport address advertised in LDP Hello messages.";
       }
     } // address-family-ipv4-augment

     grouping address-family-ipv6-augment {
       description "Augmentation to address family IPv6.";

       leaf transport-address {
         type inet:ipv6-address;
         mandatory true;
         description
           "The transport address advertised in LDP Hello messages.";
       }
     } // address-family-ipv6-augment

     grouping authentication-keychain-augment {
       description "Augmentation to authentication to add keychain.";

       leaf key-chain {
         type key-chain:key-chain-ref;
         description
           "key-chain name.";
       }
     } // authentication-keychain-augment

     grouping capability-augment {
       description "Augmentation to capability.";

       container end-of-lib {
         if-feature capability-end-of-lib;
         description
           "Configure end-of-lib capability.";
         leaf enable {
           type boolean;
           description
             "Enable end-of-lib capability.";
         }
       }
       container typed-wildcard-fec {
         if-feature capability-typed-wildcard-fec;
         description
           "Configure typed-wildcard-fec capability.";
         leaf enable {
           type boolean;
           description
             "Enable typed-wildcard-fec capability.";
         }
       }
       container upstream-label-assignment {
         if-feature capability-upstream-label-assignment;
         description
           "Configure upstream label assignment capability.";
         leaf enable {
           type boolean;
           description
             "Enable upstream label assignment.";
         }
       }
     } // capability-augment

     grouping global-augment {
       description "Augmentation to global attributes.";

       leaf igp-synchronization-delay {
         type uint16 {
           range 3..60;
         }
         units seconds;
         description
           "Sets the interval that the LDP waits before notifying the
            Interior Gateway Protocol (IGP) that label exchange is
            completed so that IGP can start advertising the normal
            metric for the link.";
       }

       uses ldp:policy-container {
         if-feature all-af-policy-config;
       }
     } // global-augment

     grouping global-forwarding-nexthop-augment {
       description
         "Augmentation to global forwarding nexthop interfaces.";

       container forwarding-nexthop {
         if-feature forwarding-nexthop-config;
         description
           "Configuration for forwarding nexthop.";

         container interfaces {
           description
             "A list of interfaces on which forwarding is disabled.";

           list interface {
             key "interface";
             description
               "List of LDP interfaces.";
             leaf interface {
               type ldp:mpls-interface-ref;
               description
                 "Interface.";
             }
             list address-family {
               key "afi";
               description
                 "Per-vrf per-af params.";
               leaf afi {
                 type ldp:ldp-address-family;
                 description
                   "Address family type value.";
               }
               container config {
                 description
                   "Configuration data.";
                 leaf ldp-disable {
                   type boolean;
                   description
                     "Disable LDP forwarding on the interface.";
                 }
               }
               container state {
                 config false;
                 description
                   "Operational state data.";
                 leaf ldp-disable {
                   type boolean;
                   description
                     "Disable LDP forwarding on the interface.";
                 }
               }
             } // address-family
           } // list interface
         } // interfaces
       } // forwarding-nexthop
     } // global-forwarding-nexthop-augment

     grouping graceful-restart-augment {
       description "Augmentation to graceful restart.";

       leaf helper-enable {
         if-feature graceful-restart-helper-mode;
         type boolean;
         description
           "Enable or disable graceful restart helper mode.";
       }
     } // graceful-restart-augment

     grouping interface-address-family-ipv4-augment {
       description "Augmentation to interface address family IPv4.";

       leaf transport-address {
         type union {
           type enumeration {
             enum "use-interface-address" {
               description
                 "Use interface address as the transport address.";
             }
           }
           type inet:ipv4-address;
         }
         description
           "IP address to be advertised as the LDP transport address.";
       }
     } // interface-address-family-ipv4-augment

     grouping interface-address-family-ipv6-augment {
       description "Augmentation to interface address family IPv6.";

       leaf transport-address {
         type union {
           type enumeration {
             enum "use-interface-address" {
               description
                 "Use interface address as the transport address.";
             }
           }
           type inet:ipv6-address;
         }
         description
           "IP address to be advertised as the LDP transport address.";
       }
     } // interface-address-family-ipv6-augment

     grouping interface-augment {
       description "Augmentation to interface.";

       uses ldp:basic-discovery-timers {
         if-feature per-interface-timer-config;
       }
       leaf igp-synchronization-delay {
         if-feature per-interface-timer-config;
         type uint16 {
           range 3..60;
         }
         units seconds;
         description
           "Sets the interval that the LDP waits before
            notifying the Interior Gateway Protocol (IGP)
            that label exchange is completed so that IGP
            can start advertising the normal metric for
            the link.";
       }
```

```
     } // interface-augment

     grouping label-policy-augment {
       description "Augmentation to graceful restart.";

       container accept {
         description
           "Label advertisement acceptance policies.";
         leaf prefix-list {
           type prefix-list-ref;
           description
             "Applies the prefix list to incoming label
              advertisements.";
         }
       } // accept
       container assign {
         if-feature policy-label-assignment-config;
         description
           "Label assignment policies";
         container independent-mode {
           description
             "Independent label policy attributes.";
           leaf prefix-list {
             type prefix-list-ref;
             description
               "Assign labels according to certain prefixes.";
           }
         } // independent-mode
         container ordered-mode {
           if-feature policy-ordered-label-config;
           description
             "Ordered label policy attributes.";
           leaf egress-prefix-list {
             type prefix-list-ref;
             description
               "Assign labels according to certain prefixes for
                egress LSR.";
           }
         } // ordered-mode
       } // assign
     } // label-policy-augment

     grouping label-policy-advertise-augment {
       description "Augmentation to graceful restart.";

       leaf prefix-list {
         type prefix-list-ref;
         description
```

```
           "Applies the prefix list to outgoing label
            advertisements.";
        }
      } // label-policy-advertise-augment

      grouping peer-af-policy-container {
        description
          "LDP policy attribute container under peer address-family.";
        container label-policy {
          description
            "Label policy attributes.";
          container advertise {
            description
              "Label advertising policies.";
            leaf prefix-list {
              type prefix-list-ref;
                description
                  "Applies the prefix list to outgoing label
                   advertisements.";
            }
          }
          container accept {
            description
              "Label advertisement acceptance policies.";
            leaf prefix-list {
              type prefix-list-ref;
              description
                "Applies the prefix list to incoming label
                 advertisements.";
            }
          } // accept
        } // label-policy
      } // peer-af-policy-container

      grouping peer-augment {
        description "Augmentation to each peer list entry.";

        leaf admin-down {
          type boolean;
          default false;
          description
            "'true' to disable the peer.";
        }

        uses peer-af-policy-container {
          if-feature all-af-policy-config;
        }
```

```
       uses ldp:graceful-restart-attributes-per-peer {
         if-feature per-peer-graceful-restart-config;
       }

       uses ldp:peer-attributes {
         if-feature per-peer-session-attributes-config;
       }
     } // peer-augment

     grouping peers-augment {
       description "Augmentation to peers container.";

       container session-downstream-on-demand {
         if-feature session-downstream-on-demand-config;
         description
           "Session downstream-on-demand attributes.";
         leaf enable {
           type boolean;
           description
             "'true' if session downstream-on-demand is enabled.";
         }
         leaf peer-list {
           type peer-list-ref;
           description
             "The name of a peer ACL.";
         }
       }
     } // peers-augment

     /*
      * Configuration and state data nodes
      */
     // Forwarding nexthop augmentation to the global tree
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global" {
       description "Graceful forwarding nexthop augmentation.";
       uses global-forwarding-nexthop-augment;
     }

     // global/address-families/ipv6
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families" {
       description "Global IPv6 augmentation.";

       container ipv6 {
         presence
           "Present if IPv6 is enabled, unless the 'enable'
            leaf is set to 'false'";
         description
           "IPv6 address family.";
         container config {
           description
             "Configuration data.";
           leaf enable {
             type boolean;
             default true;
             description
               "'true' to enable the address family.";
           }
           uses ldp:policy-container;
         }
         container state {
           config false;
           description
             "Operational state data.";
           leaf enable {
             type boolean;
             description
               "'true' to enable the address family.";
           }
           leaf label-distribution-controlmode {
             type enumeration {
               enum independent {
                 description
                 "Independent label distribution control.";
               }
               enum Ordered {
                 description
                 "Ordered Label Distribution Control.";
               }
             }
             description
               "Label distribution control mode.";
             reference
               "RFC5036: LDP Specification. Sec 2.6.";
           }

           uses ldp:policy-container;

           // ipv6 bindings
           container bindings {
             description
               "LDP address and label binding information.";
             list address {
               key "address";
               description
```

```
                 "List of address bindings.";
               leaf address {
                 type inet:ipv6-address;
                 description
                   "Binding address.";
               }
               uses ldp:binding-address-state-attributes;
             } // binding-address

             list fec-label {
               key "fec";
               description
                 "List of label bindings.";
               leaf fec {
                 type inet:ipv6-prefix;
                 description
                   "Prefix FEC.";
               }
               uses ldp:binding-label-state-attributes;
             } // fec-label
           } // bindings
         } // state
       } // ipv6
     }

     // discovery/interfaces/interface/address-families/ipv6
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:interfaces/ldp:interface/"
       + "ldp:address-families" {
       description "Interface IPv6 augmentation.";

       container ipv6 {
         presence
           "Present if IPv6 is enabled, unless the 'enable'
            leaf is set to 'false'";
         description
           "IPv6 address family.";
         container config {
           description
             "Configuration data.";
           leaf enable {
             type boolean;
             default true;
             description
               "Enable the address family on the interface.";
           }
         }
         container state {
           config false;
           description
             "Operational state data.";
           leaf enable {
             type boolean;
             description
               "Enable the address family on the interface.";
           }

           // ipv6
           list hello-adjacencies {
             key "adjacent-address";
             description "List of hello adjacencies.";

             leaf adjacent-address {
               type inet:ipv6-address;
               description
                 "Neighbor address of the hello adjacency.";
             }

             uses ldp:adjacency-state-attributes;

             leaf peer {
               type leafref {
                 path "../../../../../../../../../ldp:peers/ldp:peer/"
                   + "ldp:lsr-id";
               }
               description
                 "LDP peer from this adjacency.";
             }
           } // hello-adjacencies
         } // state
       } // ipv6
     }

     // discovery/targeted/address-families/ipv6
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:targeted/"
       + "ldp:address-families" {
       description "Targeted discovery IPv6 augmentation.";

       container ipv6 {
         presence
           "Present if IPv6 is enabled.";
         description
           "IPv6 address family.";
         container state {
           config false;
           description
             "Operational state data.";

           list hello-adjacencies {
             key "local-address adjacent-address";
             description "List of hello adjacencies.";

             leaf local-address {
               type inet:ipv6-address;
               description
                 "Local address of the hello adjacency.";
             }
             leaf adjacent-address {
               type inet:ipv6-address;
               description
                 "Neighbor address of the hello adjacency.";
             }

             uses ldp:adjacency-state-attributes;

             leaf peer {
               type leafref {
                 path "../../../../../../../../ldp:peers/ldp:peer/"
                   + "ldp:lsr-id";
               }
               description
                 "LDP peer from this adjacency.";
             }
           } // hello-adjacencies
         } // state

         list target {
           key "adjacent-address";
           description
             "Targeted discovery params.";

           leaf adjacent-address {
             type inet:ipv6-address;
             description
               "Configures a remote LDP neighbor and enables
                extended LDP discovery of the specified
                neighbor.";
           }
           container config {
             description
               "Configuration data.";
             leaf enable {
               type boolean;
               description
                 "Enable the target.";
             }
             leaf local-address {
               type inet:ipv6-address;
               description
                 "The local address.";
             }
           }
           container state {
             config false;
             description
               "Operational state data.";
             leaf enable {
               type boolean;
               description
                 "Enable the target.";
             }
             leaf local-address {
               type inet:ipv6-address;
               description
                 "The local address.";
             }
           } // state
         } // target
       } // ipv6
     }

     // /peers/peer/state/address-families/ipv6
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:state/ldp:address-families" {
       description "Peer state IPv6 augmentation.";

       container ipv6 {
         presence
           "Present if IPv6 is enabled.";
         description
           "IPv6 address family.";

         list hello-adjacencies {
           key "local-address adjacent-address";
           description "List of hello adjacencies.";

           leaf local-address {
             type inet:ipv6-address;
             description
               "Local address of the hello adjacency.";
           }
           leaf adjacent-address {
             type inet:ipv6-address;
             description
               "Neighbor address of the hello adjacency.";
           }

           uses ldp:adjacency-state-attributes;

           leaf interface {
             type ldp:mpls-interface-ref;
             description "Interface for this adjacency.";
           }
         } // hello-adjacencies
       } // ipv6
     }

     /*
      * Configuration data nodes
      */
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:config" {
       description "Graceful restart augmentation.";
       uses global-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:config/ldp:capability" {
       description "Capability augmentation.";
       uses capability-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:config/ldp:graceful-restart" {
       description "Graceful restart augmentation.";
       uses graceful-restart-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp:ipv4/ldp:config/"
       + "ldp:label-policy" {
       description "Label policy augmentation.";
       uses label-policy-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp-ext:ipv6/ldp-ext:config/"
       + "ldp-ext:label-policy" {
```

```
       description "Label policy augmentation.";
       uses label-policy-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp:ipv4/ldp:config/"
       + "ldp:label-policy/ldp:advertise" {
       description "Label policy advertise augmentation.";
       uses label-policy-advertise-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp-ext:ipv6/ldp-ext:config/"
       + "ldp-ext:label-policy/ldp-ext:advertise" {
       description "Label policy advertise augmentation.";
       uses label-policy-advertise-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp:ipv4/ldp:config" {
       description "Address family IPv4 augmentation.";
       uses address-family-ipv4-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp-ext:ipv6/ldp-ext:config" {
       description "Address family IPv4 augmentation.";
       uses address-family-ipv6-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:interfaces/ldp:interface" {
       description "Interface augmentation.";
       container config {
         description
           "Configuration data.";
         uses interface-augment;
       }
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:interfaces/ldp:interface/"
       + "ldp:address-families/ldp:ipv4/ldp:config" {
       description "Interface address family IPv4 augmentation.";
       uses interface-address-family-ipv4-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
```

```
       + "ldp:global/ldp:discovery/ldp:interfaces/ldp:interface/"
       + "ldp:address-families/ldp-ext:ipv6/ldp-ext:config" {
       description "Interface address family IPv6 augmentation.";
       uses interface-address-family-ipv6-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:targeted/ldp:config/"
       + "ldp:hello-accept" {
       description "Targeted discovery augmentation.";
       leaf neighbor-list {
         if-feature policy-targeted-discovery-config;
         type neighbor-list-ref;
         description
           "The name of a peer ACL.";
       }
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:config" {
       description "Peers augmentation.";
       uses peers-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:config/ldp:authentication/"
       + "ldp:auth-type-selection" {
       description "Peers authentication augmentation.";
       case auth-key-chain {
         uses authentication-keychain-augment;
       }
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:config" {
       description "Peer list entry augmentation.";
       uses peer-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:config/ldp:authentication/"
       + "ldp:auth-type-selection" {
       description "Peer list entry authentication augmentation.";
       case auth-key-chain {
         uses authentication-keychain-augment;
       }
     }
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:config" {
       description
         "Peer list entry augmentation to add address family.";
       container address-families {
         description
           "Per-vrf per-af params.";
         container ipv4 {
           description
             "IPv4 address family.";
           uses peer-af-policy-container;
         }
         container ipv6 {
           description
             "IPv6 address family.";
           uses peer-af-policy-container;
         } // ipv6
       } // address-family
     }

     /*
      * Operational data nodes
      */
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:state" {
       description "Graceful restart augmentation.";
       uses global-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:state/ldp:capability" {
       description "Capability augmentation.";
       uses capability-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:state/ldp:graceful-restart" {
       description "Graceful restart augmentation.";
       uses graceful-restart-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp:ipv4/ldp:state/"
       + "ldp:label-policy" {
       description "Label policy augmentation.";
       uses label-policy-augment;
     }
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp-ext:ipv6/ldp-ext:state/"
       + "ldp-ext:label-policy" {
       description "Label policy augmentation.";
       uses label-policy-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp:ipv4/ldp:state/"
       + "ldp:label-policy/ldp:advertise" {
       description "Label policy advertise augmentation.";
       uses label-policy-advertise-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp-ext:ipv6/ldp-ext:state/"
       + "ldp-ext:label-policy/ldp-ext:advertise" {
       description "Label policy advertise augmentation.";
       uses label-policy-advertise-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp:ipv4/ldp:state" {
       description "Address family IPv4 augmentation.";
       uses address-family-ipv4-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:address-families/ldp-ext:ipv6/ldp-ext:state" {
       description "Address family IPv6 augmentation.";
       uses address-family-ipv6-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:interfaces/ldp:interface/"
       + "ldp:state" {
       description "Interface augmentation.";
       uses interface-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:interfaces/ldp:interface/"
       + "ldp:address-families/ldp:ipv4/ldp:state" {
       description "Interface address family IPv4 augmentation.";
       uses interface-address-family-ipv4-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
```

```
       + "ldp:global/ldp:discovery/ldp:interfaces/ldp:interface/"
       + "ldp:address-families/ldp-ext:ipv6/ldp-ext:state" {
       description "Interface address family IPv6 augmentation.";
       uses interface-address-family-ipv6-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:global/ldp:discovery/ldp:targeted/ldp:state/"
       + "ldp:hello-accept" {
       description "Targeted discovery augmentation.";
       leaf neighbor-list {
         if-feature policy-targeted-discovery-config;
         type neighbor-list-ref;
         description
           "The name of a peer ACL.";
       }
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:state" {
       description "Peers augmentation.";
       uses peers-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:state/ldp:authentication/"
       + "ldp:auth-type-selection" {
       description "Peers authentication augmentation.";
       case auth-key-chain {
         uses authentication-keychain-augment;
       }
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:state" {
       description "Peer list entry augmentation.";
       uses peer-augment;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:state/ldp:authentication/"
       + "ldp:auth-type-selection" {
       description "Peer list entry authentication augmentation.";
       case auth-key-chain {
         uses authentication-keychain-augment;
       }
     }
     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:state/ldp:address-families/ldp:ipv4" {
       description
         "Peer list entry IPv4 augmentation.";
       uses peer-af-policy-container;
     }

     augment "/rt:routing/rt:control-plane-protocols/ldp:mpls-ldp/"
       + "ldp:peers/ldp:peer/ldp:state/ldp:address-families/"
       + "ldp-ext:ipv6" {
       description
         "Peer list entry IPv6 augmentation.";
       uses peer-af-policy-container;
     }

     /*
      * RPCs
      */

     /*
      * Notifications
      */
   }

   <CODE ENDS>
```

Figure 17

10. Security Considerations

The configuration, state, action and notification data defined using YANG data models in this document are likely to be accessed via the protocols such as NETCONF [RFC6241] etc.

Hence, YANG implementations MUST comply with the security requirements specified in section 15 of [RFC6020]. Additionally, NETCONF implementations MUST comply with the security requirements specified in sections 2.2, 2.3 and 9 of [RFC6241] as well as section 3.7 of [RFC6536].

11. IANA Considerations

This document does not extend LDP base protocol specifiction and hence there are no IANA considerations.

Note to the RFC Editor: Please remove IANA section before the publication.

12. Acknowledgments

The authors would like to acknowledge Eddie Chami, Nagendra Kumar, Mannan Venkatesan, Pavan Beeram for their contribution to this document. We also acknowledge Ladislav Lhotka for his useful comments as the YANG Doctor.

13. References

13.1. Normative References

[I-D.ietf-mpls-base-yang]
Raza, K., Gandhi, R., Liu, X., Beeram, V., Saad, T., Bryskin, I., Chen, X., Jones, R., and B. Wen, "A YANG Data Model for MPLS Base", draft-ietf-mpls-base-yang-05 (work in progress), July 2017.

[I-D.ietf-mpls-mldp-yang]
Raza, K., Krishnaswamy, S., Liu, X., Esale, S., Chen, X., and j. jefftant@gmail.com, "YANG Data Model for MPLS mLDP", draft-ietf-mpls-mldp-yang-01 (work in progress), March 2017.

[I-D.rtgyangdt-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "Network Instance Model", draft-rtgyangdt-rtgwg-ni-model-00 (work in progress), May 2016.

[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>.

[RFC3478] Leelanivas, M., Rekhter, Y., and R. Aggarwal, "Graceful Restart Mechanism for Label Distribution Protocol", RFC 3478, DOI 10.17487/RFC3478, February 2003, <http://www.rfc-editor.org/info/rfc3478>.

[RFC5036] Andersson, L., Ed., Minei, I., Ed., and B. Thomas, Ed., "LDP Specification", RFC 5036, DOI 10.17487/RFC5036, October 2007, <http://www.rfc-editor.org/info/rfc5036>.

[RFC5331] Aggarwal, R., Rekhter, Y., and E. Rosen, "MPLS Upstream Label Assignment and Context-Specific Label Space", RFC 5331, DOI 10.17487/RFC5331, August 2008, <http://www.rfc-editor.org/info/rfc5331>.

[RFC5561] Thomas, B., Raza, K., Aggarwal, S., Aggarwal, R., and JL. Le Roux, "LDP Capabilities", RFC 5561, DOI 10.17487/RFC5561, July 2009, <http://www.rfc-editor.org/info/rfc5561>.

[RFC5918] Asati, R., Minei, I., and B. Thomas, "Label Distribution Protocol (LDP) 'Typed Wildcard' Forward Equivalence Class (FEC)", RFC 5918, DOI 10.17487/RFC5918, August 2010, <http://www.rfc-editor.org/info/rfc5918>.

[RFC5919] Asati, R., Mohapatra, P., Chen, E., and B. Thomas, "Signaling LDP Label Advertisement Completion", RFC 5919, DOI 10.17487/RFC5919, August 2010, <http://www.rfc-editor.org/info/rfc5919>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6389] Aggarwal, R. and JL. Le Roux, "MPLS Upstream Label Assignment for LDP", RFC 6389, DOI 10.17487/RFC6389, November 2011, <http://www.rfc-editor.org/info/rfc6389>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

[RFC7552] Asati, R., Pignataro, C., Raza, K., Manral, V., and R. Papneja, "Updates to LDP for IPv6", RFC 7552, DOI 10.17487/RFC7552, June 2015, <http://www.rfc-editor.org/info/rfc7552>.

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <http://www.rfc-editor.org/info/rfc8022>.

13.2. Informative References

[I-D.ietf-rtgwg-policy-model]
Shaikh, A., Shakir, R., D'Souza, K., and C. Chase, "Routing Policy Configuration Model for Service Provider Networks", draft-ietf-rtgwg-policy-model-01 (work in progress), April 2016.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <http://www.rfc-editor.org/info/rfc4364>.

[RFC7307] Zhao, Q., Raza, K., Zhou, C., Fang, L., Li, L., and D. King, "LDP Extensions for Multi-Topology", RFC 7307, DOI 10.17487/RFC7307, July 2014, <http://www.rfc-editor.org/info/rfc7307>.

Appendix A. Additional Contributors

Reshad Rahman
Cisco Systems Inc.
Email: rrahman@cisco.com

Stephane Litkowski
Orange.
Email: stephane.litkowski@orange.com

Danial Johari
Cisco Systems Inc.
Email: dajohari@cisco.com

Authors' Addresses

Kamran Raza
Cisco Systems, Inc.
Email: skraza@cisco.com

Rajiv Asati
Cisco Systems, Inc.
Email: rajiva@cisco.com

Xufeng Liu
Jabil
Email: xufeng_liu@jabil.com

Jeff Tantsura
Email: jefftant.ietf@gmail.com

Santosh Esale
Juniper Networks
Email: sesale@juniper.net

Xia Chen
Huawei Technologies
Email: jescia.chenxia@huawei.com

Loa Andersson
Huawei Technologies
Email: loa@pi.nu

Himanshu Shah
Ciena Corporation
Email: hshah@ciena.com

Matthew Bocci
Nokia
Email: matthew.bocci@nokia.com

BFD Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2018

G. Mirsky
ZTE Corp.

October 23, 2017

BFD in Demand Mode over Point-to-Point MPLS LSP
draft-mirsky-bfd-mpls-demand-02

Abstract

This document describes procedures for using Bidirectional Forwarding Detection (BFD) in Demand mode to detect data plane failures in Multiprotocol Label Switching (MPLS) point-to-point Label Switched Paths.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC5884] defined use of the Asynchronous method of Bidirectional Detection (BFD) [RFC5880] to monitor and detect failures in data path of a Multiprotocol Label Switching (MPLS) Label Switched Path (LSP). Use of the Demand mode, also specified in [RFC5880], has not been defined so far. This document describes procedures for using the Demand mode of BFD protocol to detect data plane failures in MPLS point-to-point (p2p) LSPs.

2. Conventions used in this document

2.1. Terminology

MPLS: Multiprotocol Label Switching

LSP: Label Switched Path

LER: Label switching Edge Router

BFD: Bidirectional Forwarding Detection

p2p: Point-to-Point

2.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Use of the BFD Demand Mode

[RFC5880] defines that the Demand mode MAY be:

- o asymmetric, i.e. used in one direction of a BFD session;
- o switched to and from without bringing BFD session to Down state through using a Poll Sequence.

For the case of BFD over MPLS LSP, ingress Label switching Edge Router (LER) is usually acts as Active BFD peer and egress LER acts as Passive BFD peer. The Active peer bootstraps the BFD session by using LSP ping. Once the BFD session is in Up state the ingress LER that supports this specification MUST switch to the Demand mode by setting Demand (D) bit in its Control packet and initiating a Poll Sequence. If the egress LER supports this specification it MUST respond with the Final (F) bit set in its BFD Control packet sent to the ingress LER and ceases further transmission of periodic BFD control packets to the ingress LER.

In this state BFD peers MAY remain as long as the egress LER is in Up state. The ingress LER MAY check liveness of the egress LER by setting Poll flag. The egress LER will respond by transmitting BFD control packet with the Final flag set. If the ingress LER doesn't receive BFD packet with the Final flag from its peer after predetermined period of time, default wait time recommended 1 second, the ingress MAY transmit another packet with the Poll flag set. If ingress doesn't receive BFD control packet with the Final flag set in response to three consecutive packets with Poll flag, it MAY declare the BFD peer non-responsive and change state of the BFD session to Down state.

If the Detection timer at the egress LER expires it MUST send BFD Control packet to the ingress LER with the Poll (P) bit set, Status (Sta) field set to Down value, and the Diagnostic (Diag) field set to Control Detection Time Expired value. The egress LER sends these Control packets to the ingress LER at the rate of one per second until either it receives the valid for this BFD session control packet with the Final (F) bit set from the ingress LER or the defect condition clears and the BFD session state reaches Up state at the egress LER.

The ingress LER transmits BFD Control packets over the MPLS LSP with the Demand (D) flag set at negotiated interval per [RFC5880], the greater of bfd.DesiredMinTxInterval and bfd.RemoteMinRxInterval, until it receives the valid BFD packet from the egress LER with the Poll (P) bit and the Diagnostic (Diag) field value Control Detection Time Expired. Receprion of such BFD control packet by the ingress

LER indicates that the monitored LSP has a failure and sending BFD control packet with Final flag set to acknowledge failure indication is likely to fail. Instead, the ingress LER transmits the BFD Control packet to the egress LER over the IP network with:

- destination IP address MUST be set to the destination IP address of the LSP Ping Echo request message [RFC8029];
- destination UDP port set to 4784 [RFC5883];
- Final (F) flag in BFD control packet MUST be set;
- Demand (D) flag in BFD control packet MUST be cleared.

The ingress LER changes the state of the BFD session to Down and changes rate of BFD Control packets transmission to one packet per second. The ingress LER in Down mode changes to Asynchronous mode until the BFD session comes to Up state once again. Then the ingress LER switches to the Demand mode.

4. IANA Considerations

TBD

5. Security Considerations

This document does not introduce new security aspects but inherits all security considerations from [RFC5880], [RFC5884], [RFC7726], [RFC8029], and [RFC6425].

6. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <https://www.rfc-editor.org/info/rfc5880>.

[RFC5883] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD) for Multihop Paths", RFC 5883, DOI 10.17487/RFC5883, June 2010, <https://www.rfc-editor.org/info/rfc5883>.

[RFC5884] Aggarwal, R., Kompella, K., Nadeau, T., and G. Swallow, "Bidirectional Forwarding Detection (BFD) for MPLS Label Switched Paths (LSPs)", RFC 5884, DOI 10.17487/RFC5884, June 2010, <https://www.rfc-editor.org/info/rfc5884>.

[RFC6425] Saxena, S., Ed., Swallow, G., Ali, Z., Farrel, A., Yasukawa, S., and T. Nadeau, "Detecting Data-Plane Failures in Point-to-Multipoint MPLS - Extensions to LSP Ping", RFC 6425, DOI 10.17487/RFC6425, November 2011, <https://www.rfc-editor.org/info/rfc6425>.

[RFC7726] Govindan, V., Rajaraman, K., Mirsky, G., Akiya, N., and S. Aldrin, "Clarifying Procedures for Establishing BFD Sessions for MPLS Label Switched Paths (LSPs)", RFC 7726, DOI 10.17487/RFC7726, January 2016, <https://www.rfc-editor.org/info/rfc7726>.

[RFC8029] Kompella, K., Swallow, G., Pignataro, C., Ed., Kumar, N., Aldrin, S., and M. Chen, "Detecting Multiprotocol Label Switched (MPLS) Data-Plane Failures", RFC 8029, DOI 10.17487/RFC8029, March 2017, <https://www.rfc-editor.org/info/rfc8029>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Appendix A. Acknowledgements

TBD

Author's Address

Greg Mirsky
ZTE Corp.

Email: gregimirsky@gmail.com

MPLS Working Group
Internet-Draft
Updates: 5884 (if approved)
Intended status: Standards Track
Expires: April 21, 2018

G. Mirsky
Y. Zhao
ZTE Corporation

October 18, 2017

Clarifying Use of LSP Ping to Bootstrap BFD over MPLS LSP
draft-mirsky-mpls-bfd-bootstrap-clarify-00

Abstract

This document, if approved, updates RFC 5884 by clarifying procedures for using MPLS LSP ping to bootstrap Bidirectional Forwarding Detection (BFD) over MPLS Label Switch Path.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC5884] defines how LSP Ping [RFC8029] uses BFD Discriminator TLV to bootstrap Bidirectional Forwarding Detection (BFD) session over MPLS Label Switch Path (LSP). Implementation and operational experiences suggest that two aspects of using LSP ping to bootstrap BFD session can benefit from clarification. This document updates [RFC5884] in use of Return mode field in MPLS LSP echo request message and use of BFD Discriminator TLV in MPLS LSP echo reply.

2. Conventions used in this document

2.1. Terminology

MPLS: Multiprotocol Label Switching

LSP: Label Switched Path

BFD: Bidirectional Forwarding Detection

2.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Use of Return Mode Field

[RFC5884] does not define the value to be used for the Return mode field [RFC8029] when LSP ping is used to bootstrap a BFD session of MPLS LSP. When LSP echo request is being used to detect defects in MPLS data plane and verify consistency between the control plane and the data plane echo reply is needed to confirm the correct state, provide the positive acknowledgment. But when LSP echo request is being used to bootstrap BFD session, then the positive acknowledgement, according to [RFC5884] is provided by the egress transmitting BFD control message. Thus LSP echo reply is not required to bootstrap BFD session and hence the Return mode field in echo request message SHOULD be set to 1 (Do not reply) [RFC8029] when LSP echo request used to bootstrap BFD session.

4. Use of BFD Discriminator TLV in LSP Echo Reply

[RFC5884] in section 6 defines that echo reply by the egress LSR to BFD bootstrapping echo request MAY include BFD Discriminator TLV with locally assigned discriminator value for the BFD session. But the [RFC5884] does not define how the ingress LSR may use the returned value. From practical point, as discussed in Section 3, the returned value is not useful since the egress is required to send the BFD control message right after successfully validating the FEC and before sending echo reply message. Secondly, identifying the corresponding BFD session at ingress without returning its discriminator presents unnecessary challenge for the implementation. Thus the egress LSR SHOULD NOT include BFD Discriminator TLV if sending echo reply to BFD bootstrapping echo request.

5. IANA Considerations

This document does not require any action by IANA. This section may be removed.

6. Security Considerations

This document does not introduce new security aspects but inherits all security considerations from [RFC5880], [RFC5884], [RFC8029].

7. Acknowledgements

TBA

8. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <https://www.rfc-editor.org/info/rfc5880>.

[RFC5884] Aggarwal, R., Kompella, K., Nadeau, T., and G. Swallow, "Bidirectional Forwarding Detection (BFD) for MPLS Label Switched Paths (LSPs)", RFC 5884, DOI 10.17487/RFC5884, June 2010, <https://www.rfc-editor.org/info/rfc5884>.

[RFC8029] Kompella, K., Swallow, G., Pignataro, C., Ed., Kumar, N., Aldrin, S., and M. Chen, "Detecting Multiprotocol Label Switched (MPLS) Data-Plane Failures", RFC 8029, DOI 10.17487/RFC8029, March 2017, <https://www.rfc-editor.org/info/rfc8029>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Authors' Addresses

Greg Mirsky
ZTE Corporation

Email: gregimirsky@gmail.com

Yanhua Zhao
ZTE Corporation

Email: zhao.yanhua3@zte.com.cn

SPRING Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 27, 2018

G. Mirsky
ZTE Corp.
J. Tantsura
Individual
I. Varlashkin
Google
M. Chen
Huawei

October 24, 2017

Bidirectional Forwarding Detection (BFD) in Segment Routing Networks Using MPLS Dataplane
draft-mirsky-spring-bfd-02

Abstract

Segment Routing architecture leverages the paradigm of source routing. It can be realized in the Multiprotocol Label Switching (MPLS) network without any change to the data plane. A segment is encoded as an MPLS label and an ordered list of segments is encoded as a stack of labels. Bidirectional Forwarding Detection (BFD) is expected to monitor any kind of paths between systems. This document defines how to use Label Switched Path Ping to bootstrap and control path in reverse direction of a BFD session on the Segment Routing static MPLS tunnel.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC5880], [RFC5881], and [RFC5883] established the Bidirectional Forwarding Detection (BFD) protocol for IP networks. [RFC5884] and [RFC7726] set rules of using BFD Asynchronous mode over Multiprotocol Label Switching (MPLS) Label Switched Path (LSP). These latter standards implicitly assume that the egress BFD peer, which is the egress Label Edge Router (LER), will use the shortest path route regardless of the path the ingress LER uses to send BFD control packets towards it.

This document defines use of LSP Ping for Segment Routing networks over MPLS dataplane [I-D.ietf-mpls-spring-lsp-ping] to bootstrap and control path of a BFD session from the egress to ingress LER using static MPLS tunnel.

1.1. Conventions used in this document

1.1.1. Terminology

BFD: Bidirectional Forwarding Detection

FEC: Forwarding Equivalence Class

MPLS: Multiprotocol Label Switching

LSP: Label Switching Path

LER: Label Edge Router

1.1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Bootstrapping BFD session over Segment Routed tunnel

As demonstrated in [I-D.ietf-mpls-spring-lsp-ping] introduction of Segment Routing network domains with an MPLS data plane requires three new sub-TLVs that MAY be used with Target Forwarding Equivalence Class (FEC) TLV. Section 6.1 addresses use of the new sub-TLVs in Target FEC TLV in LSP ping and LSP traceroute. For the case of LSP ping the [I-D.ietf-mpls-spring-lsp-ping] states that:

> Initiator MUST include FEC(s) corresponding to the destination segment.
>
> Initiator, i.e. ingress LSR, MAY include FECs corresponding to some or all of segments imposed in the label stack by the ingress LSR to communicate the segments traversed.

It has been noted in [RFC5884] that a BFD session monitors for defects particular <MPLS LSP, FEC> tuple. [RFC7726] clarified how to establish and operate multiple BFD sessions for the same <MPLS LSP, FEC> tuple. Because only ingress edge router is aware of the SR-based explicit route the egress edge router can associate the LSP ping with BFD Discriminator TLV with only one of the FECs it advertised for the particular segment. Thus this document clarifies that:

> When LSP Ping is used to bootstrap a BFD session the FEC corresponding to the destination segment to be associated with the BFD session MUST be as the very last sub-TLV in the Target FEC TLV.

Encapsulation of a BFD Control packet in Segment Routing network with MPLS dataplane MUST follow Section 7 [RFC5884] when IP/UDP header used and MUST follow Section 3.4 [RFC6428] without IP/UDP header being used.

3. Use BFD Reverse Path TLV over SDN-provisioned Segment Routed MPLS Tunnel

For BFD over MPLS LSP case, per [RFC5884], egress LER MAY send BFD control packet to the ingress LER either over IP network or an MPLS LSP. Similarly, for the case of BFD over p2p segment tunnel with MPLS data plane, the ingress LER MAY route BFD control packet over IP network, as described in [RFC5883], or transmit over a segment tunnel, as described in Section 7 [RFC5884]. In some cases there may be a need to direct egress BFD peer to use specific path for the reverse direction of the BFD session by using the BFD Reverse Path TLV [I-D.ietf-mpls-bfd-directed]. For the case of MPLS dataplane, Segment Routing Architecture [I-D.ietf-spring-segment-routing] explains that "a segment is encoded as an MPLS label. An ordered list of segments is encoded as a stack of labels." YANG Data Model for MPLS Static LSPs [I-D.ietf-mpls-static-yang] models outgoing MPLS labels to be imposed as leaf-list [RFC6020], i.e., as array of rt-types:mpls-label [I-D.ietf-rtgwg-routing-types] Following on that, this document defines Segment Routing Static MPLS Tunnel sub-TLV that MAY be used with the BFD Reverse Path TLV [I-D.ietf-mpls-bfd-directed]. The format of the sub-TLV is presented in Figure 1. BFD Reverse TLV MAY include zero or one SR Static MPLS Tunnel 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  SegRouting MPLS sub-TLV Type |          Length               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          Label Entry 1                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          Label Entry 2                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                                                               ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          Label Entry N                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Segment Routing Static MPLS Tunnel sub-TLV

The Segment Routing Tunnel sub-TLV Type is two octets in length, and has a value of TBD (to be assigned by IANA as requested in Section 5).

The egress LSR MUST use the Value field as label stack for BFD control packets for the BFD session identified by the source IP address of the MPLS LSP Ping packet and the value in the BFD Discriminator TLV. Label Entries MUST be in network order.

As in [I-D.ietf-mpls-bfd-directed], empty BFD Reverse TLV requires the egress BFD peer switch the reverse path of the BFD session, specified by BFD Discriminator TLV, to the path selected based on locally defined policy. If more than one SR Static MPLS Tunnel sub-TLV is present, then the egress BFD peer MUST send Echo Reply with Return Code field set to "Too Many TLVs Detected" Table 2.

The Segment Routing Tunnel sub-TLV MAY be used in Reply Path TLV defined in [RFC7110]

4. BFD Reverse Path TLV over Segment Routed MPLS Tunnel with Dynamic Control Plane

When Segment Routed domain with MPLS data plane uses distributed tunnel computation BFD Reverse Path TLV MAY use Target FEC sub-TLVs defined in [I-D.ietf-mpls-spring-lsp-ping].

5. IANA Considerations

5.1. Segment Routing Static MPLS Tunnel sub-TLV

The IANA is requested to assign new sub-TLV type from "Multiprotocol Label Switching Architecture (MPLS) Label Switched Paths (LSPs) Ping Parameters - TLVs" registry, "Sub-TLVs for TLV Types 1, 16, and 21" sub-registry.

Value	Description	Reference
X (TBD1)	Segment Routing Static MPLS Tunnel sub-TLV	This document

Table 1: New Segment Routing Tunnel sub-TLV

5.2. Return Code

The IANA is requested to assign a new Return Code value from the "Multi-Protocol Label Switching (MPLS) Label Switched Paths (LSPs) Ping Parameters" registry, "Return Codes" sub-registry, as follows using a Standards Action value.

Value	Description	Reference
X (TBD2)	Too Many TLVs Detected.	This document

Table 2: New Return Code

6. Security Considerations

Security considerations discussed in [RFC5880], [RFC5884], [RFC7726], and [RFC8029] apply to this document.

7. Acknowledgements

TBD

8. References

8.1. Normative References

[I-D.ietf-mpls-bfd-directed]
Mirsky, G., Tantsura, J., Varlashkin, I., and M. Chen, "Bidirectional Forwarding Detection (BFD) Directed Return Path", draft-ietf-mpls-bfd-directed-07 (work in progress), June 2017.

[I-D.ietf-mpls-spring-lsp-ping]
Kumar, N., Pignataro, C., Swallow, G., Akiya, N., Kini, S., and M. Chen, "Label Switched Path (LSP) Ping/ Traceroute for Segment Routing IGP Prefix and Adjacency SIDs with MPLS Data-plane", draft-ietf-mpls-spring-lsp-ping-13 (work in progress), October 2017.

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-12 (work in progress), June 2017.

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

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <https://www.rfc-editor.org/info/rfc5880>.

[RFC5881] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop)", RFC 5881, DOI 10.17487/RFC5881, June 2010, <https://www.rfc-editor.org/info/rfc5881>.

[RFC5883] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD) for Multihop Paths", RFC 5883, DOI 10.17487/RFC5883, June 2010, <https://www.rfc-editor.org/info/rfc5883>.

[RFC5884] Aggarwal, R., Kompella, K., Nadeau, T., and G. Swallow, "Bidirectional Forwarding Detection (BFD) for MPLS Label Switched Paths (LSPs)", RFC 5884, DOI 10.17487/RFC5884, June 2010, <https://www.rfc-editor.org/info/rfc5884>.

[RFC6428] Allan, D., Ed., Swallow, G., Ed., and J. Drake, Ed., "Proactive Connectivity Verification, Continuity Check, and Remote Defect Indication for the MPLS Transport Profile", RFC 6428, DOI 10.17487/RFC6428, November 2011, <https://www.rfc-editor.org/info/rfc6428>.

[RFC7110] Chen, M., Cao, W., Ning, S., Jounay, F., and S. Delord, "Return Path Specified Label Switched Path (LSP) Ping", RFC 7110, DOI 10.17487/RFC7110, January 2014, <https://www.rfc-editor.org/info/rfc7110>.

[RFC7726] Govindan, V., Rajaraman, K., Mirsky, G., Akiya, N., and S. Aldrin, "Clarifying Procedures for Establishing BFD Sessions for MPLS Label Switched Paths (LSPs)", RFC 7726, DOI 10.17487/RFC7726, January 2016, <https://www.rfc-editor.org/info/rfc7726>.

[RFC8029] Kompella, K., Swallow, G., Pignataro, C., Ed., Kumar, N., Aldrin, S., and M. Chen, "Detecting Multiprotocol Label Switched (MPLS) Data-Plane Failures", RFC 8029, DOI 10.17487/RFC8029, March 2017, <https://www.rfc-editor.org/info/rfc8029>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

8.2. Informative References

[I-D.ietf-mpls-static-yang]
Saad, T., Raza, K., Gandhi, R., Liu, X., Beeram, V., Shah, H., Bryskin, I., Chen, X., Jones, R., and B. Wen, "A YANG Data Model for MPLS Static LSPs", draft-ietf-mpls-static-yang-04 (work in progress), July 2017.

[I-D.ietf-rtgwg-routing-types]
Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Routing Area Common YANG Data Types", draft-ietf-rtgwg-routing-types-17 (work in progress), October 2017.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

Authors' Addresses

Greg Mirsky
ZTE Corp.

Email: gregimirsky@gmail.com

Jeff Tantsura
Individual

Email: jefftant.ietf@gmail.com

Ilya Varlashkin
Google

Email: Ilya@nobulus.com

Mach(Guoyi) Chen
Huawei

Email: mach.chen@huawei.com

MPLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2017

X. Xu
S. Bryant
Huawei
H. Assarpour
Broadcom
H. Shah
Ciena
L. Contreras
Telefonica I+D
D. Bernier
Bell Canada
J. Tantsura
Individual
S. Ma
Juniper
M. Vigoureux
Nokia
June 29, 2017

Service Chaining using Unified Source Routing Instructions
draft-xu-mpls-service-chaining-03

Abstract

Source Packet Routing in Networking (SPRING) WG is developing an MPLS source routing mechanism. The MPLS source routing mechanism can be leveraged to realize a unified source routing instruction which works across both IPv4 and IPv6 underlays in addition to the MPLS underlay. This document describes how to leverage the unified source routing instruction to realize a transport-independent service function chaining by encoding the service function path information or service function chain information as an MPLS label stack.

Requirements Language

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 RFC 2119 [RFC2119].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

When applying a particular Service Function Chain (SFC) [RFC7665] to the traffic selected by a service classifier, the traffic need to be steered through an ordered set of Service Functions (SF) in the network. This ordered set of SFs in the network indicates the Service Function Path (SFP) associated with the above SFC. In order to steer the selected traffic through the required ordered list of SFs, the service classifier needs to attach information to the packet specifying exactly which Service Function Forwarders (SFFs) and which SFs are to be visited by traffic), the SFC, or the partially specified SFP which is in between the former two extremes.

The Source Packet Routing in Networking (SPRING) WG is developing an MPLS source routing mechanism which can be used to steer traffic through an ordered set of routers (i.e., an explicit path) and instruct nodes on that path to execute specific operations on the packet. By leveraging the MPLS source routing mechanism, [I-D.xu-mpls-unified-source-routing-instruction] describes a unified source routing instruction which works across both IPv4 and IPv6 underlays in addition to the MPLS underlay. This document describes how to leverage the unified source routing instruction to realize a transport-independent service function chaining by encoding the service function path information or service function chain information as an MPLS label stack.

2. Terminology

This memo makes use of the terms defined in [I-D.ietf-spring-segment-routing-mpls], [I-D.xu-mpls-unified-source-routing-instruction] and [RFC7665].

3. Solution Description

```
           +---------------------------------------------- ----+
           |               MPLS SPRING Networks                |
           |            +---------+       +---------+          |
           |            |   SF1   |       |   SF2   |          |
           |            +----+----+       +----+----+          |
           |               ^ | |(3)          ^ | |(6)          |
           |       (1)  (2)| | V     (4)  (5)| | V     (7)     |
      +----+-----+ ---> +----+----+ ----> +----+----+ ---> +---+---+
      |Classifier+------+  SFF1   +-------+  SFF2   +------+   D   |
      +----------+      +---------+       +---------+      +---+---+
           |                                                   |
           +---------------------------------------------------+
```

Figure 1: Service Function Chaining in MPLS-SPRING Networks

As shown in Figure 1, SFF1 and SFF2 are two MPLS-SPRING-capable nodes. They are also SFFs, each with one SF attached. In addition, they have allocated and advertised MPLS labels for their locally attached SFs. For example, SFF1 allocates and advertises a label (i.e., L(SF1)) for SF1 while SFF2 allocates and advertises a label (i.e., L(SF2)) for SF2. These labels, which are used to indicate SFs are referred to as SF labels. To encode the SFP information as an

MPLS label stack, local MPLS labels are allocated from SFFs' (e.g., SFF1 in Figure 1) label spaces to identify their locally attached SFs (e.g., SF1 in Figure 1), whilst the SFFs are identified by either nodal SIDs or adjacency SIDs depending on how strictly the network path needs to be specified. In addition, assume node SIDs for SFF1 and SFF2 are L(SFF1) and L(SFF2) respectively. In contrast, to encode the SFC information by an MPLS label stack, those SF labels MUST be domain-wide unique MPLS labels.

Now assume a given traffic flow destined for destination D is selected by the service classifier to go through a particular SFC (i.e., SF1-> SF2) before reaching its final destination D. Section 3.1 and 3.2 describe approaches of leveraging the MPLS- based source routing mechanisms to realize the service function chaining by encoding the SFP information within an MPLS label stack and by encoding the SFC information within an MPLS label stack respectively. Since the encoding of the partially specified SFP is just a simple combination of the encoding of the SFP and the encoding of the SFC, this document would not describe how to encode the partially specified SFP anymore.

3.1. Encoding SFP Information by an MPLS Label Stack

```
           +----------------------------------------------- ----+
           |               MPLS SPRING Networks                 |
           |            +---------+       +---------+           |
           |            |   SF1   |       |   SF2   |           |
           |            +----+----+       +----+----+           |
           |  +---------+    |                 |    +---------+ |
           |  | L(SFF2) |    |                 |    |Pkt to D | |
           |  +---------+    |                 |    +---------+ |
           |  | L(SF2)  |    |                 |                |
           |  +---------+    |               ^ | |              |
           |  |Pkt to D | ^  | |             | | |              |
           |  +---------+ |  | |          (5)| | |(6)           |
           |           (2)|  | |(3)          | | V              |
           |       (1)    |  | V     (4)       |      (7)       |
      +----+-----+ ---> +----+----+ ----> +----+----+ --->  +---+---+
      |Classifier+------+  SFF1   +-------+  SFF2   +-------+   D   |
      +----------+      +---------+       +---------+       +---+---+
           |    +---------+      +---------+        +---------+ |
           |    | L(SFF1) |      | L(SFF2) |        |Pkt to D | |
           |    +---------+      +---------+        +---------+ |
           |    | L(SF1)  |      | L(SF2)  |                    |
           |    +---------+      +---------+                    |
           |    | L(SFF2) |      |Pkt to D |                    |
           |    +---------+      +---------+                    |
           |    | L(SF2)  |                                     |
           |    +---------+                                     |
           |    |Pkt to D |                                     |
           |    +---------+                                     |
           +----------------------------------------------------+
```

Figure 2: Packet Walk in MPLS underlay

As shown in Figure 2, since the selected packet needs to travel through an SFC (i.e., SF1->SF2), the service classifier would attach a segment list of (i.e., SID(SFF1)->SID(SF1)->SID(SFF2)-> SID(SF2)) which indicates the corresponding SFP to the packet. This segment list is represented by an MPLS label stack. To some extent, the MPLS label stack here could be looked as a specific implementation of the SFC encapsulation used for containing the SFP information [RFC7665]. When the encapsulated packet arrives at SFF1, SFF1 would know which SF should be performed according to the top label (i.e., SID (SF1)) of the received MPLS packet. We first consider the case where SF1 is an encapsulation aware SF, i.e., it understands how to process a packet with a pre-pended MPLS label stack. In this case the packet would be sent to SF1 by SFF1 with the label stack SID(SFF2)-> SID(SF2). SF1 would perform the required service function on the received MPLS packet where the payload is constrained to be an IP packet, and the SF needs to process both IPv4 and IPv6 packets (note that the SF would use the first nibble of the MPLS payload to identify the payload type). After the MPLS packet is returned from SF1, SFF1 would send it to SFF2 according to the top label (i.e., SID (SFF2)).

If SF1 is a legacy SF, i.e. one that is unable to process the MPLS label stack, the remaining MPLS label stack (i.e., SID(SFF2)->SID(SF2)) MUST be saved and stripped from the packet before sending the packet to SF1. When the packet is returned from SF1, SFF1 would re-impose the MPLS label stack which had been previously stripped and then send the packet to SFF2 according to the current top label (i.e., SID (SFF2)). As for how to associate the corresponding MPLS label stack with the packets returned from legacy SFs, those mechanisms as described in [I-D.song-sfc-legacy-sf-mapping] could be considered.

When the encapsulated packet arrives at SFF2, SFF2 would perform the similar action to that described above.

As shown in Figure 3, if there is no MPLS LSP towards the next node segment (i.e., the next SFF identified by the current top label), the corresponding IP-based tunnel for MPLS (e.g., MPLS-in-IP/GRE tunnel [RFC4023], MPLS-in-UDP tunnel [RFC7510] or MPLS-in-L2TPv3 tunnel [RFC4817]) would be used instead, according to the unified source routing instruction as described in [I-D.xu-mpls-unified-source-routing-instruction].

```
           +----------------------------------------------- ----+
           |                      IP Networks                   |
           |            +---------+       +---------+           |
           |            |   SF1   |       |   SF2   |           |
           |            +----+----+       +----+----+           |
           |  +---------+    |                 |    +---------+ |
           |  | L(SFF2) |    |                 |    |Pkt to D | |
           |  +---------+    |                 |    +---------+ |
           |  | L(SF2)  |    |                 |                |
           |  +---------+    |               ^ | |              |
           |  |Pkt to D | ^  | |             | | |              |
           |  +---------+ |  | |          (5)| | |(6)           |
           |           (2)|  | |(3)          | | V              |
           |       (1)    |  | V    (4)        |      (7)       |
      +----+-----+ ---> +----+----+ ----> +----+----+ --->  +---+---+
      |Classifier+------+  SFF1   +-------+  SFF2   +-------+   D   |
      +----------+      +---------+       +---------+       +---+---+
           |    +---------+      +---------+                    |
           |    |IP Tunnel|      |IP Tunnel|        +---------+ |
           |    |to SFF1  |      | to SFF2 |        |Pkt to D | |
           |    +---------+      +---------+        +---------+ |
           |    | L(SF1)  |      | L(SF2)  |                    |
           |    +---------+      +---------+                    |
           |    | L(SFF2) |      |Pkt to D |                    |
           |    +---------+      +---------+                    |
           |    | L(SF2)  |                                     |
           |    +---------+                                     |
           |    |Pkt to D |                                     |
           |    +---------+                                     |
           +----------------------------------------------------+
```

Figure 3: Packet Walk in IP underlay

Since the transport (i.e., the underlay) could be IPv4, IPv6 or even MPLS networks, the above approach of encoding the SFP information by an MPLS label stack is fully transport-independent which is one of the major requirements for the SFC encapsulation [RFC7665].

3.2. Encoding SFC Information by an MPLS Label Stack

Since the selected packet needs to travel through an SFC (i.e., SF1->SF2), the service classifier would attach an MPLS label stack (i.e., L(SF1)->L(SF2)) which indicates that SFC to the packet. Since it’s known to the service classifier that SFF1 is attached with an instance of SF1, the service classifier would therefore send the MPLS encapsulated packet through either an MPLS LSP tunnel or an IP-based tunnel towards SFF1 (as shown in Figure 4 and 5 respectively). When the MPLS encapsulated packet arrives at SFF1, SFF1 would know which SF should be performed according to the current top label (i.e., L(SF1)). Similarly, SFF1 would send the packet returned from SF1 to SFF2 through either an MPLS LSP tunnel or an IP-based tunnel towards SFF2 since it's known to SFF1 that SFF2 is attached with an instance of SF2. When the encapsulated packet arrives at SFF2, SFF2 would do the similar action as what has been done by SFF1. Since the transport (i.e., the underlay) could be IPv4, IPv6 or even MPLS networks, the above approach of encoding the SFC information by an MPLS label stack is fully transport-independent which is one of the major requirements for the SFC encapsulation [RFC7665].

```
           +----------------------------------------------- ----+
           |                    MPLS Networks                   |
           |            +---------+       +---------+           |
           |            |   SF1   |       |   SF2   |           |
           |            +----+----+       +----+----+           |
           |                 |                 |    +---------+ |
           |                 |                 |    |Pkt to D | |
           |  +---------+    |                 |    +---------+ |
           |  | L(SF2)  |    |                 |                |
           |  +---------+    |               ^ | |              |
           |  |Pkt to D | ^  | |             | | |              |
           |  +---------+ |  | |          (5)| | |(6)           |
           |           (2)|  | |(3)          | | V              |
           |       (1)    |  | V     (4)       |      (7)       |
      +----+-----+ ---> +----+----+ ----> +----+----+ --->  +---+---+
      |Classifier+------+  SFF1   +-------+  SFF2   +-------+   D   |
      +----------+      +---------+       +---------+       +---+---+
           |    +---------+      +---------+        +---------+ |
           |    | L(SFF1) |      | L(SFF2) |        |Pkt to D | |
           |    +---------+      +---------+        +---------+ |
           |    | L(SF1)  |      | L(SF2)  |                    |
           |    +---------+      +---------+                    |
           |    | L(SF2)  |      |Pkt to D |                    |
           |    +---------+      +---------+                    |
           |    |Pkt to D |                                     |
           |    +---------+                                     |
           +----------------------------------------------------+
```
Figure 4: Packet Walk in MPLS underlay

```
           +----------------------------------------------- ----+
           |                    IP Networks                   |
           |            +---------+       +---------+           |
           |            |   SF1   |       |   SF2   |           |
           |            +----+----+       +----+----+           |
           |                 |                 |    +---------+ |
           |                 |                 |    |Pkt to D | |
           |  +---------+    |                 |    +---------+ |
           |  | L(SF2)  |    |                 |                |
           |  +---------+    |               ^ | |              |
           |  |Pkt to D | ^  | |             | | |              |
           |  +---------+ |  | |          (5)| | |(6)           |
           |           (2)|  | |(3)          | | V              |
           |       (1)    |  | V     (4)       |      (7)       |
      +----+-----+ ---> +----+----+ ----> +----+----+ --->  +---+---+
      |Classifier+------+  SFF1   +-------+  SFF2   +-------+   D   |
      +----------+      +---------+       +---------+       +---+---+
           |    +---------+      +---------+                    |
           |    |IP Tunnel|      |IP Tunnel|        +---------+ |
           |    |to SFF1  |      | to SFF2 |        |Pkt to D | |
           |    +---------+      +---------+        +---------+ |
           |    | L(SF1)  |      | L(SF2)  |                    |
           |    +---------+      +---------+                    |
           |    | L(SF2)  |      |Pkt to D |                    |
           |    +---------+      +---------+                    |
           |    |Pkt to D |                                     |
           |    +---------+                                     |
           +----------------------------------------------------+
```

Figure 5: Packet Walk in IP underlay

3.3. How to Contain Metadata within an MPLS Packet

Since the MPLS encapsulation has no explicit protocol identifier field to indicate the protocol type of the MPLS payload, how to indicate the presence of metadata (i.e., the NSH which is only used as a metadata containner) in an MPLS packet is a potential issue to be addressed. One possible way to address the above issue is: SFFs allocate two different labels for a given SF, one indicates the presence of NSH while the other indicates the absence of NSH. This approach has no change to the current MPLS architecture but it would require more than one label binding for a given SF. Another possible way is to introduce a protocol identifier field within the MPLS packet as described in [I-D.xu-mpls-payload-protocol-identifier].

More details about how to contain metadata within an MPLS packet would be considered in the future version of this draft.

4. Acknowledgements

The authors would like to thank Loa Andersson, Andrew G. Malis, Adrian Farrel, Alexander Vainshtein and Joel M. Halpern for their valuable comments and suggestions on the document.

5. IANA Considerations

This document makes no request of IANA.

6. Security Considerations

It is fundamental to the SFC design that the classifier is a trusted resource which determines the processing that the packet will be subject to, including for example the firewall. It is also fundamental to the SPRING design that packets are routed through the network using the path specified by the node imposing the SIDs. Where an SF is not encapsulation aware the packet may exist as an IP packet, however this is an intrinsic part of the SFC design which needs to define how a packet is protected in that environment. Where a tunnel is used to link two non-MPLS domains, the tunnel design needs to specify how it is secured. Thus the secutity vulnerabilities are addressed in the underlying technologies used by this design, which itself does not introduce any new security vulnerabilities.

7. References

7.1. 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>.

7.2. Informative References

[I-D.ietf-sfc-nsh]
Quinn, P. and U. Elzur, "Network Service Header", draft-ietf-sfc-nsh-12 (work in progress), February 2017.

[I-D.ietf-spring-segment-routing-mpls]
Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with MPLS data plane", draft-ietf-spring-segment-routing-mpls-10 (work in progress), June 2017.

[I-D.song-sfc-legacy-sf-mapping]
Song, H., You, J., Yong, L., Jiang, Y., Dunbar, L., Bouthors, N., and D. Dolson, "SFC Header Mapping for Legacy SF", draft-song-sfc-legacy-sf-mapping-08 (work in progress), September 2016.

[I-D.xu-mpls-payload-protocol-identifier]
Xu, X., "MPLS Payload Protocol Identifier", draft-xu-mpls-payload-protocol-identifier-02 (work in progress), December 2016.

[I-D.xu-mpls-unified-source-routing-instruction]
Xu, X., Bryant, S., Raszuk, R., Chunduri, U., Contreras, L., Jalil, L., Assarpour, H., Velde, G., Tantsura, J., and S. Ma, "Unified Source Routing Instruction using MPLS Label Stack", draft-xu-mpls-unified-source-routing-instruction-02 (work in progress), June 2017.

[RFC4023] Worster, T., Rekhter, Y., and E. Rosen, Ed., "Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)", RFC 4023, DOI 10.17487/RFC4023, March 2005, <http://www.rfc-editor.org/info/rfc4023>.

[RFC4817] Townsley, M., Pignataro, C., Wainner, S., Seely, T., and J. Young, "Encapsulation of MPLS over Layer 2 Tunneling Protocol Version 3", RFC 4817, DOI 10.17487/RFC4817, March 2007, <http://www.rfc-editor.org/info/rfc4817>.

[RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black, "Encapsulating MPLS in UDP", RFC 7510, DOI 10.17487/RFC7510, April 2015, <http://www.rfc-editor.org/info/rfc7510>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <http://www.rfc-editor.org/info/rfc7665>.

Authors' Addresses

Xiaohu Xu
Huawei

Email: xuxiaohu@huawei.com

Stewart Bryant
Huawei

Email: stewart.bryant@gmail.com

Hamid Assarpour
Broadcom

Email: hamid.assarpour@broadcom.com

Himanshu Shah
Ciena

Email: hshah@ciena.com

Luis M. Contreras
Telefonica I+D
Ronda de la Comunicacion, s/n
Sur-3 building, 3rd floor
Madrid, 28050
Spain

Email: luismiguel.contrerasmurillo@telefonica.com
URI: http://people.tid.es/LuisM.Contreras/

Daniel Bernier
Bell Canada

Email: daniel.bernier@bell.ca

Jeff Tantsura
Individual

Email: jefftant@gmail.com

Shaowen Ma
Juniper

Email: mashaowen@gmail.com

Martin Vigoureux
Nokia

Email: martin.vigoureux@nokia.com