ï»¿

Internet Engineering Task Force (IETF)
Request for Comments: 8662
Category: Standards Track
ISSN: 2070-1721

S. Kini

K. Kompella
Juniper

S. Sivabalan
Cisco

S. Litkowski
Orange

R. Shakir
Google

J. Tantsura
Apstra, Inc.

December 2019

Entropy Label for Source Packet Routing in Networking (SPRING) Tunnels

Abstract

Segment Routing (SR) leverages the source-routing paradigm. A node steers a packet through an ordered list of instructions, called segments. Segment Routing can be applied to the Multiprotocol Label Switching (MPLS) data plane. Entropy labels (ELs) are used in MPLS to improve load-balancing. This document examines and describes how ELs are to be applied to Segment Routing MPLS.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8662.

Table of Contents

1. Introduction

Segment Routing [RFC8402] is based on source-routed tunnels to steer a packet along a particular path. This path is encoded as an ordered list of segments. When applied to the MPLS data plane [RFC8660], each segment is an LSP (Label Switched Path) with an associated MPLS label value. Hence, label stacking is used to represent the ordered list of segments, and the label stack associated with an SR tunnel can be seen as nested LSPs (LSP hierarchy) in the MPLS architecture.

Using label stacking to encode the list of segments has implications on the label stack depth.

Traffic load-balancing over ECMP (Equal-Cost Multipath) or LAGs (Link Aggregation Groups) is usually based on a hashing function. The local node that performs the load-balancing is required to read some header fields in the incoming packets and then compute a hash based on those fields. The result of the hash is finally mapped to a list of outgoing next hops. The hashing technique is required to perform a per-flow load-balancing and thus, prevents packet misordering. For IP traffic, the usual fields that are hashed are the source address, the destination address, the protocol type, and, if provided by the upper layer, the source port and destination port.

The MPLS architecture brings some challenges when an LSR (Label Switching Router) tries to look up at header fields. An LSR needs be able to look up at header fields that are beyond the MPLS label stack while the MPLS header does not provide any information about the upper-layer protocol. An LSR must perform a deeper inspection compared to an ingress router, which could be challenging for some hardware. Entropy labels (ELs) [RFC6790] are used in the MPLS data plane to provide entropy for load-balancing. The idea behind the entropy label is that the ingress router computes a hash based on several fields from a given packet and places the result in an additional label named "entropy label". Then, this entropy label can be used as part of the hash keys used by an LSR. Using the entropy label as part of the hash keys reduces the need for deep packet inspection in the LSR while keeping a good level of entropy in the load-balancing. When the entropy label is used, the keys used in the hashing functions are still a local configuration matter, and an LSR may use solely the entropy label or a combination of multiple fields from the incoming packet.

When using LSP hierarchies, there are implications on how [RFC6790] should be applied. The current document addresses the case where a hierarchy is created at a single LSR as required by Segment Routing.

A use case requiring load-balancing with SR is given in Section 3. A recommended solution is described in Section 7 keeping in consideration the limitations of implementations when applying [RFC6790] to deeper label stacks. Options that were considered to arrive at the recommended solution are documented for historical purposes in Section 10.

1.1. 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.

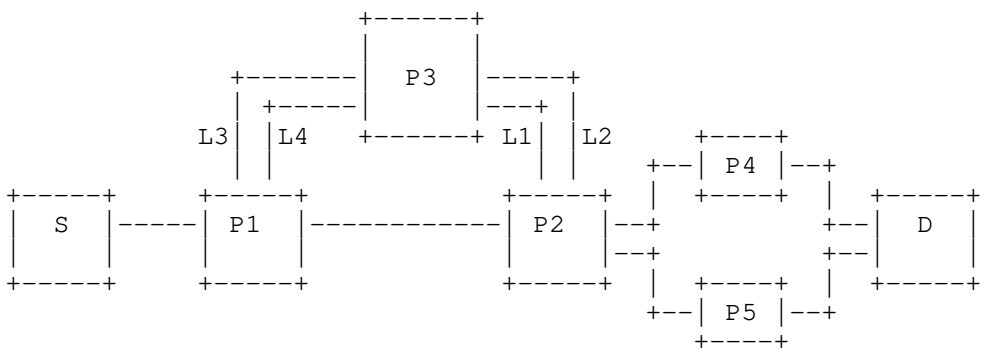
2. Abbreviations and Terminology

Adj-SID	Adjacency Segment Identifier
ECMP	Equal-Cost Multipath
EL	Entropy Label
ELI	Entropy Label Indicator
ELC	Entropy Label Capability
ERLD	Entropy Readable Label Depth
FEC	Forwarding Equivalence Class
LAG	Link Aggregation Group
LSP	Label Switched Path
LSR	Label Switching Router
MPLS	Multiprotocol Label Switching
MSD	Maximum SID Depth
Node SID	Node Segment Identifier
OAM	Operations, Administration, and Maintenance
RLD	Readable Label Depth
SID	Segment Identifier
SPT	Shortest Path Tree
SR	Segment Routing
SRGB	Segment Routing Global Block
VPN	Virtual Private Network

3. Use Case Requiring Multipath Load-Balancing

Traffic engineering is one of the applications of MPLS and is also a requirement for Segment Routing [RFC7855]. Consider the topology shown in Figure 1. The LSR S requires data to be sent to LSR D along a traffic-engineered path that goes over the link L1. Good load-balancing is also required across equal-cost paths (including parallel links). To steer traffic along a path that crosses link L1, the label stack that LSR S creates consists of a label to the Node SID of LSR P3 stacked over the label for the Adj-SID (Adjacency Segment Identifier) of link L1 and that in turn is stacked over the label to the Node SID of LSR D. For simplicity, lets assume that all LSRs use the same label space for Segment Routing (as a reminder, it

is called the SRGB, Segment Routing Global Block). Let L_N-Px denote the label to be used to reach the Node SID of LSR Px. Let L_A-Ln denote the label used for the Adj-SID for link Ln. In our example, the LSR S must use the label stack <L_N-P3, L_A-L1, L_N-D>. However, to achieve good load-balancing over the equal-cost paths P2-P4-D, P2-P5-D, and the parallel links L3 and L4, a mechanism such as entropy labels [RFC6790] should be adapted for Segment Routing. Indeed, the Source Packet Routing in Networking (SPRING) architecture with the MPLS data plane [RFC8660] uses nested MPLS LSPs composing the source-routed label stack.



Key:
S = Source LSR
D = Destination LSR
P1, P2, P3, P4, P5 = Transit LSRs
L1, L2, L3, L4 = Links

Figure 1: Traffic-Engineering Use Case

An MPLS node may have limitations in the number of labels it can push. It may also have a limitation in the number of labels it can inspect when looking for hash keys during load-balancing. While the entropy label is normally inserted at the bottom of the transport tunnel, this may prevent an LSR from taking into account the EL in its load-balancing function if the EL is too deep in the stack. In a Segment Routing environment, it is important to define the considerations that need to be taken into account when inserting an EL. Multiple ways to apply entropy labels were considered and are documented in Section 10 along with their trade-offs. A recommended solution is described in Section 7.

4. Entropy Readable Label Depth

The Entropy Readable Label Depth (ERLD) is defined as the number of labels a router can both:

a. Read in an MPLS packet received on its incoming interface(s) (starting from the top of the stack).

b. Use in its load-balancing function.

The ERLD means that the router will perform load-balancing using the EL if the EL is placed within the first ERLD labels.

A router capable of reading N labels but not using an EL located within those N labels MUST consider its ERLD to be 0.

In a distributed switching architecture, each line card may have a different capability in terms of ERLD. For simplicity, an implementation MAY use the minimum ERLD of all line cards as the ERLD value for the system.

There may also be a case where a router has a fast switching path (handled by an Application-Specific Integrated Circuit, or ASIC, or network processor) and a slow switching path (handled by a CPU) with a different ERLD for each switching path. Again, for simplicity's sake, an implementation MAY use the minimum ERLD as the ERLD value for the system.

The drawback of using a single ERLD for a system lower than the capability of one or more specific components is that it may increase the number of ELI/ELs inserted. This leads to an increase of the label stack size and may have an impact on the capability of the ingress node to push this label stack.

Examples:

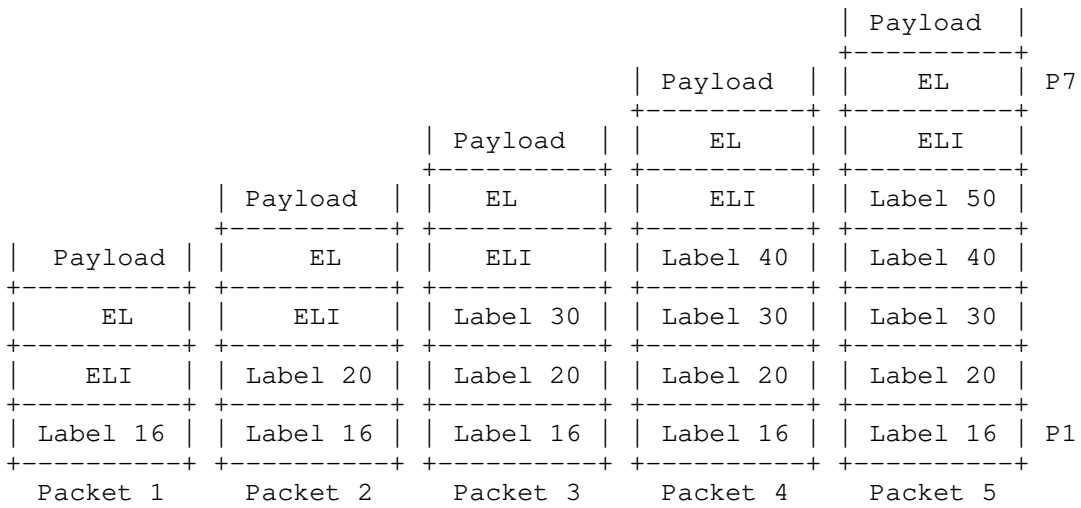


Figure 2: Label Stacks with ELI/EL

In Figure 2, we consider the displayed packets received on a router interface. We consider also a single ERLD value for the router.

* If the router has an ERLD of 3, it will be able to load-balance Packet 1 displayed in Figure 2 using the EL as part of the load-balancing keys. The ERLD value of 3 means that the router can read and take into account the entropy label for load-balancing if it is placed between position 1 (top of the MPLS label stack) and position 3.

* If the router has an ERLD of 5, it will be able to load-balance Packets 1 to 3 in Figure 2 using the EL as part of the load-balancing keys. Packets 4 and 5 have the EL placed at a position greater than 5, so the router is not able to read it and use it as part of the load-balancing keys.

* If the router has an ERLD of 10, it will be able to load-balance all the packets displayed in Figure 2 using the EL as part of the load-balancing keys.

To allow an efficient load-balancing based on entropy labels, a router running SPRING SHOULD advertise its ERLD (or ERLDs), so all the other SPRING routers in the network are aware of its capability. How this advertisement is done is outside the scope of this document (see Section 7.2.1 for potential approaches).

To advertise an ERLD value, a SPRING router:

* MUST be entropy label capable and, as a consequence, MUST apply the data-plane procedures defined in [RFC6790].

* MUST be able to read an ELI/EL, which is located within its ERLD value.

* MUST take into account an EL within the first ERLD labels in its load-balancing function.

5. Maximum SID Depth

The Maximum SID Depth defines the maximum number of labels that a particular node can impose on a packet. This can include any kind of labels (service, entropy, transport, etc.). In an MPLS network, the MSD is a limit of the head-end of an SR tunnel or a Binding SID

anchor node that performs imposition of additional labels on an existing label stack.

Depending on the number of MPLS operations (POP, SWAP, etc.) to be performed before the PUSH, the MSD can vary due to hardware or software limitations. As for the ERLD, different MSD limits can exist within a single node based on the line-card types used in a distributed switching system. Thus, the MSD is a per link and/or per-node property.

An external controller can be used to program a label stack on a particular node. This node SHOULD advertise its MSD to the controller in order to let the controller know the maximum label stack depth of the path computed that is supported on the head-end. How this advertisement is done is outside the scope of this document. ([RFC8476], [RFC8491], and [MSD-BGP] provide examples of advertisement of the MSD.) As the controller does not have the knowledge of the entire label stack to be pushed by the node, in addition to the MSD value, the node SHOULD advertise the type of the MSD. For instance, the MSD value can represent the limit for pushing transport labels only while in reality the node can push an additional service label. As another example, the MSD value can represent the full limit of the node including all label types (transport, service, entropy, etc.). This gives the ability for the controller to program a label stack while leaving room for the local node to add more labels (e.g., service, entropy, etc.) without reaching the hardware/software limit. If the node does not provide the meaning of the MSD value, the controller could program an LSP using a number of labels equal to the full limit of the node. When receiving this label stack from the controller, the ingress node may not be able to add any service (L2VPN, L3VPN, EVPN, etc.) label on top of this label stack. The consequence could be for the ingress node to drop service packets that should have been forwarded over the LSP.

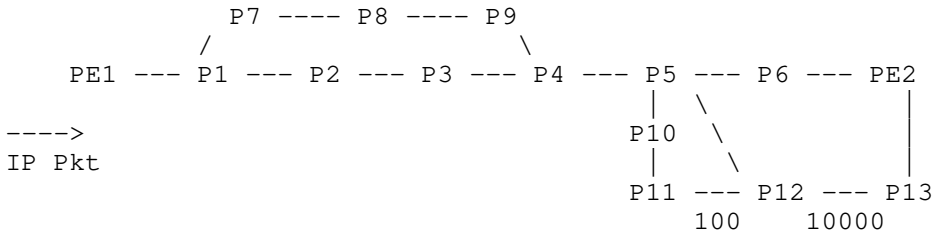


Figure 3: Topology Illustrating Label Stack Reduction

In Figure 3, an IP packet comes into the MPLS network at PE1. All metrics are considered equal to 1 except P12-P13, which is 10000, and P11-P12, which is 100. PE1 wants to steer the traffic using a SPRING path to PE2 along PE1 -> P1 -> P7 -> P8 -> P9 -> P4 -> P5 -> P10 -> P11 -> P12 -> P13 -> PE2. By using Adj-SIDs only, PE1 (acting as an ingress LSR, also known as an I-LSR) will be required to push 10 labels on the IP packet received and thus, requires an MSD of 10. If the IP packet should be carried over an MPLS service like a regular layer 3 VPN, an additional service label should be imposed requiring an MSD of 11 for PE1. In addition, if PE1 wants to insert an ELI/EL for load-balancing purposes, PE1 will need to push 13 labels on the IP packet requiring an MSD of 13.

In the SPRING architecture, Node SIDs or Binding SIDs can be used to reduce the label stack size. As an example, to steer the traffic on the same path as before, PE1 could use the following label stack: <Node_P9, Node_P5, Binding_P5, Node_PE2>. In this example, we consider a combination of Node SIDs and a Binding SID advertised by P5 that will stitch the traffic along the path P10 -> P11 -> P12 -> P13. The instruction associated with the Binding SID at P5 is thus to swap Binding_P5 to Adj_P12-P13 and then push <Adj_P11-P12, Node_P11>. P5 acts as a stitching node that pushes additional labels on an existing label stack; P5's MSD needs also to be taken into account and may limit the number of labels that can be imposed.

6. LSP Stitching Using the Binding SID

The Binding SID allows binding a segment identifier to an existing LSP. As examples, the Binding SID can represent an RSVP-TE tunnel, an LDP path (through the Mapping Server Advertisement), or a SPRING path. Each tail-end router of an MPLS LSP associated with a Binding SID has its own entropy label capability. The entropy label capability of the associated LSP is advertised in the control-plane protocol used to signal the LSP.

In Figure 4, we consider that:

* P6, PE2, P10, P11, P12, and P13 are pure LDP routers.

* PE1, P1, P2, P3, P4, P7, P8, and P9 are pure SPRING routers.

* P5 is running SPRING and LDP.

* P5 acts as a Mapping Server and advertises Prefix-SIDs for the LDP FECs: an index value of 20 is used for PE2.

* All SPRING routers use an SRGB of [1000, 1999].

* P6 advertises label 20 for the PE2 FEC.

* Traffic from PE1 to PE2 uses the shortest path.

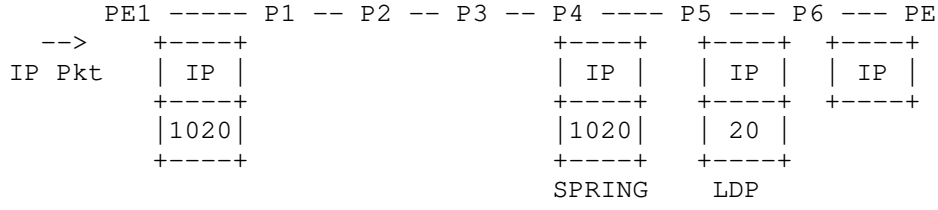


Figure 4: Example Illustrating Need for ELC Propagation

In terms of packet forwarding, by learning the Mapping Server Advertisement from P5, PE1 imposes a label 1020 to an IP packet destined to PE2. SPRING routers along the shortest path to PE2 will switch the traffic until it reaches P5. P5 will perform the LSP stitching by swapping the SPRING label 1020 to the LDP label 20 advertised by the next hop P6. P6 will finally forward the packet using the LDP label towards PE2.

PE1 cannot push an ELI/EL for the Binding SID without knowing that the tail end of the LSP associated with the binding (PE2) is entropy label capable.

To accommodate the mix of signaling protocols involved during the stitching, the entropy label capability SHOULD be propagated between the signaling domains. Each Binding SID SHOULD have its own entropy label capability that MUST be inherited from the entropy label capability of the associated LSP. If the router advertising the Binding SID does not know the ELC state of the target FEC, it MUST NOT set the ELC for the Binding SID. An ingress node MUST NOT push an ELI/EL associated with a Binding SID unless this Binding SID has the entropy label capability. How the entropy label capability is advertised for a Binding SID is outside the scope of this document (see Section 7.2.1 for potential approaches).

In our example, if PE2 is LDP entropy label capable, it will add the entropy label capability in its LDP advertisement. When P5 receives the FEC/label binding for PE2, it learns about the ELC and can set the ELC in the Mapping Server Advertisement. Thus, PE1 learns about the ELC of PE2 and may push an ELI/EL associated with the Binding SID.

The proposed solution only works if the SPRING router advertising the Binding SID is also performing the data-plane LSP stitching. In our example, if the Mapping Server function is hosted on P8 instead of

P5, P8 does not know about the ELC state of PE2's LDP FEC. As a consequence, it does not set the ELC for the associated Binding SID.

7. Insertion of Entropy Labels for SPRING Path

7.1. Overview

The solution described in this section follows the data-plane processing defined in [RFC6790]. Within a SPRING path, a node may be ingress, egress, transit (regarding the entropy label processing described in [RFC6790]), or it can be any combination of those. For example:

* The ingress node of a SPRING domain can be an ingress node from an entropy label perspective.

* Any LSR terminating a segment of the SPRING path is an egress node (because it terminates the segment) but can also be a transit node if the SPRING path is not terminated because there is a subsequent SPRING MPLS label in the stack.

* Any LSR processing a Binding SID may be a transit node and an ingress node (because it may push additional labels when processing the Binding SID).

As described earlier, an LSR may have a limitation (the ERLD) on the depth of the label stack that it can read and process in order to do multipath load-balancing based on entropy labels.

If an EL does not occur within the ERLD of an LSR in the label stack of an MPLS packet that it receives, then it would lead to poor load-balancing at that LSR. Hence, an ELI/EL pair must be within the ERLD of the LSR in order for the LSR to use the EL during load-balancing.

Adding a single ELI/EL pair for the entire SPRING path can also lead to poor load-balancing as well because the ELI/EL may not occur within the ERLD of some LSR on the path (if too deep) or may not be present in the stack when it reaches some LSRs (if it is too shallow).

In order for the EL to occur within the ERLD of LSRs along the path corresponding to a SPRING label stack, multiple <ELI, EL> pairs MAY be inserted in this label stack.

The insertion of an ELI/EL MUST occur only with a SPRING label advertised by an LSR that advertised an ERLD (the LSR is entropy label capable) or with a SPRING label associated with a Binding SID that has the ELC set.

The ELs among multiple <ELI, EL> pairs inserted in the stack MAY be the same or different. The LSR that inserts <ELI, EL> pairs can have limitations on the number of such pairs that it can insert and also the depth at which it can insert them. If, due to limitations, the inserted ELs are at positions such that an LSR along the path receives an MPLS packet without an EL in the label stack within that LSR's ERLD, then the load-balancing performed by that LSR would be poor. An implementation MAY consider multiple criteria when inserting <ELI, EL> pairs.

7.1.1. Example 1: The Ingress Node Has a Sufficient MSD

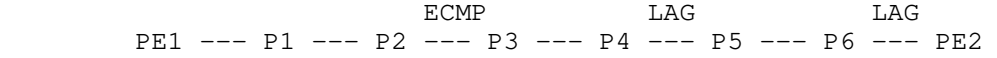


Figure 5: Accommodating MSD Limitations

In Figure 5, PE1 wants to forward some MPLS VPN traffic over an explicit path to PE2 resulting in the following label stack to be pushed onto the received IP header: <Adj_P1P2, Adj_set_P2P3, Adj_P3P4, Adj_P4P5, Adj_P5P6, Adj_P6PE2, VPN_label>. PE1 is limited to push a maximum of 11 labels (MSD=11). P2, P3, and P6 have an ERLD

of 3 while others have an ERLD of 10.

PE1 can only add two ELI/EL pairs in the label stack due to its MSD limitation. It should insert them strategically to benefit load-balancing along the longest part of the path.

PE1 can take into account multiple parameters when inserting ELs; as examples:

* The ERLD value advertised by transit nodes.

* The requirement of load-balancing for a particular label value.

* Any service provider preference: favor beginning of the path or end of the path.

In Figure 5, a good strategy may be to use the following stack <Adj_P1P2, Adj_set_P2P3, ELI1, EL1, Adj_P3P4, Adj_P4P5, Adj_P5P6, Adj_P6PE2, ELI2, EL2, VPN_label>. The original stack requests P2 to forward based on an L3 adjacency-set that will require load-balancing. Therefore, it is important to ensure that P2 can load-balance correctly. As P2 has a limited ERLD of 3, an ELI/EL must be inserted just after the label that P2 will use to forward. On the path to PE2, P3 has also a limited ERLD, but P3 will forward based on a regular adjacency segment that may not require load-balancing. Therefore, it does not seem important to ensure that P3 can do load-balancing despite its limited ERLD. The next nodes along the forwarding path have a high ERLD that does not cause any issue, except P6. Moreover, P6 is using some LAGs to PE2 and so is expected to load-balance. It becomes important to insert a new ELI/EL just after the P6 forwarding label.

In the case above, the ingress node was able to support a sufficient MSD to ensure end-to-end load-balancing while taking into account the path attributes. However, there might be cases where the ingress node may not have the necessary label imposition capacity.

7.1.2. Example 2: The Ingress Node Does Not Have a Sufficient MSD

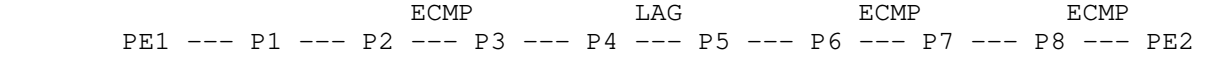


Figure 6: MSD Considerations

In Figure 6, PE1 wants to forward MPLS VPN traffic over an explicit path to PE2 resulting in the following label stack to be pushed onto the IP header: <Adj_P1P2, Adj_set_P2P3, Adj_P3P4, Adj_P4P5, Adj_P5P6, Adj_set_P6P7, Adj_P7P8; Adj_set_P8PE2, VPN_label>. PE1 is limited to push a maximum of 11 labels. P2, P3, and P6 have an ERLD of 3 while others have an ERLD of 15.

Using a similar strategy as the previous case may lead to a dilemma, as PE1 can only push a single ELI/EL while we may need a minimum of three to load-balance the end-to-end path. An optimized stack that would enable end-to-end load-balancing may be: <Adj_P1P2, Adj_set_P2P3, ELI1, EL1, Adj_P3P4, Adj_P4P5, Adj_P5P6, Adj_set_P6P7, ELI2, EL2, Adj_P7P8, Adj_set_P8PE2, ELI3, EL3, VPN_label>.

A decision needs to be taken to favor some part of the path for load-balancing considering that load-balancing may not work on the other parts. A service provider may decide to place the ELI/EL after the P6 forwarding label as it will allow P4 and P6 to load-balance. Placing the ELI/EL at the bottom of the stack is also a possibility enabling load-balancing for P4 and P8.

7.2. Considerations for the Placement of Entropy Labels

The sample cases described in the previous section showed that ELI/EL placement when the maximum number of labels to be pushed is limited is not an easy decision, and multiple criteria may be taken into account.

This section describes some considerations that an implementation MAY take into account when placing ELI/ELs. This list of criteria is not considered exhaustive and an implementation MAY take into account additional criteria or tiebreakers that are not documented here. As the insertion of ELI/ELs is performed by the ingress node, having ingress nodes that do not use the same criteria does not cause an interoperability issue. However, from a network design and operation perspective, it is better to have all ingress routers using the same criteria.

An implementation SHOULD try to maximize the possibility of load-balancing along the path by inserting an ELI/EL where multiple equal-cost paths are available and minimize the number of ELI/ELs that need to be inserted. In case of a trade-off, an implementation SHOULD provide flexibility to the operator to select the criteria to be considered when placing ELI/ELs or specify a subobjective for optimization.

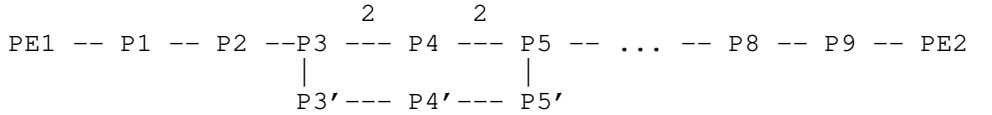


Figure 7: MSD Trade-Offs

Figure 7 will be used as reference in the following subsections. All metrics are equal to 1 except P3-P4 and P4-P5, which have a metric 2. We consider the MSD of nodes to be the full limit of label imposition (including service labels, entropy labels, and transport labels).

7.2.1. ERLD Value

As mentioned in Section 7.1, the ERLD value is an important parameter to consider when inserting an ELI/EL. If an ELI/EL does not fall within the ERLD of a node on the path, the node will not be able to load-balance the traffic efficiently.

The ERLD value can be advertised via protocols, and those extensions are described in separate documents (for instance, [ISIS-ELC] and [OSPF-ELC]).

Let's consider a path from PE1 to PE2 using the following stack pushed by PE1: <Adj_P1P2, Node_P9, Adj_P9PE2, Service_label>.

Using the ERLD as an input parameter can help to minimize the number of required ELI/EL pairs to be inserted. An ERLD value must be retrieved for each SPRING label in the label stack.

For a label bound to an adjacency segment, the ERLD is the ERLD of the node that has advertised the adjacency segment. In the example above, the ERLD associated with Adj_P1P2 would be the ERLD of router P1, as P1 will perform the forwarding based on the Adj_P1P2 label.

For a label bound to a node segment, multiple strategies MAY be implemented. An implementation MAY try to evaluate the minimum ERLD value along the node segment path. If an implementation cannot find the minimum ERLD along the path of the segment or does not support the computation of the minimum ERLD, it SHOULD instead use the ERLD of the tail-end node. Using the ERLD of the tail end of the node segment mimics the behavior of [RFC6790] where the ingress takes only care of the egress of the LSP. In the example above, if the implementation supports computation of minimum ERLD along the path, the ERLD associated with label Node_P9 would be the minimum ERLD between nodes {P2,P3,P4 ..., P8}. If the implementation does not support the computation of minimum ERLD, it will consider the ERLD of P9 (tail-end node of Node_P9 SID). While providing the more optimal ELI/EL placement, evaluating the minimum ERLD increases the complexity of ELI/EL insertion. As the path to the Node SID may change over time, a recomputation of the minimum ERLD is required for each topology change. This recomputation may require the positions

of the ELI/ELs to change.

For a label bound to a Binding Segment, if the Binding Segment describes a path, an implementation MAY also try to evaluate the minimum ERLD along this path. If the implementation cannot find the minimum ERLD along the path of the segment or does not support this evaluation, it SHOULD instead use the ERLD of the node advertising the Binding SID. As for the node segment, evaluating the minimum ERLD adds complexity in the ELI/EL insertion process.

7.2.2. Segment Type

Depending on the type of segment a particular label is bound to, an implementation can deduce that this particular label will be subject to load-balancing on the path.

7.2.2.1. Node SID

An MPLS label bound to a Node SID represents a path that may cross multiple hops. Load-balancing may be needed on the node starting this path but also on any node along the path.

In Figure 7, let's consider a path from PE1 to PE2 using the following stack pushed by PE1: <Adj_P1P2, Node_P9, Adj_P9PE2, Service_label>.

If, for example, PE1 is limited to push 6 labels, it can add a single ELI/EL within the label stack. An operator may want to favor a placement that would allow load-balancing along the Node SID path. In Figure 7, P3, which is along the Node SID path, requires load-balancing between two equal-cost paths.

An implementation MAY try to evaluate if load-balancing is really required within a node segment path. This could be done by running an additional SPT (Shortest Path Tree) computation and analyzing of the node segment path to prevent a node segment that does not really require load-balancing from being preferred when placing ELI/ELs. Such inspection may be time consuming for implementations and without a 100% guarantee, as a node segment path may use LAGs that are invisible to the IP topology. As a simpler approach, an implementation MAY consider that a label bound to a Node SID will be subject to load-balancing and require an ELI/EL.

7.2.2.2. Adjacency-Set SID

An adjacency-set is an Adj-SID that refers to a set of adjacencies. When an adjacency-set segment is used within a label stack, an implementation can deduce that load-balancing is expected at the node that advertised this adjacency segment. An implementation MAY favor the insertion of an ELI/EL after the Adj-SID representing an adjacency-set.

7.2.2.3. Adjacency SID Representing a Single IP Link

When an adjacency segment representing a single IP link is used within a label stack, an implementation can deduce that load-balancing may not be expected at the node that advertised this adjacency segment.

An implementation MAY NOT place an ELI/EL after a regular Adj-SID in order to favor the insertion of ELI/ELs following other segments.

Readers should note that an adjacency segment representing a single IP link may require load-balancing. This is the case when a LAG (L2 bundle) is implemented between two IP nodes and the L2 bundle SR extensions [RFC8668] are not implemented. In such a case, it could be useful to insert an ELI/EL in a readable position for the LSR advertising the label associated with the adjacency segment. To communicate the requirement for load-balancing for a particular Adjacency SID to ingress nodes, a user can enforce the use of the L2 bundle SR extensions defined in [RFC8668] or can declare the single

adjacency as an adjacency-set.

7.2.2.4. Adjacency SID Representing a Single Link within an L2 Bundle

When the L2 bundle SR extensions [RFC8668] are used, adjacency segments may be advertised for each member of the bundle. In this case, an implementation can deduce that load-balancing is not expected on the LSR advertising this segment and MAY NOT insert an ELI/EL after the corresponding label.

7.2.2.5. Adjacency SID Representing an L2 Bundle

When the L2 bundle SR extensions [RFC8668] are used, an adjacency segment may be advertised to represent the bundle. In this case, an implementation can deduce that load-balancing is expected on the LSR advertising this segment and MAY insert an ELI/EL after the corresponding label.

7.2.3. Maximizing Number of LSRs That Will Load-Balance

When placing ELI/ELs, an implementation MAY optimize the number of LSRs that both need to load-balance (i.e., have ECMPs) and that will be able to perform load-balancing (i.e., the EL is within their ERLD).

Let's consider a path from PE1 to PE2 using the following stack pushed by PE1: <Adj_P1P2, Node_P9, Adj_P9PE2, Service_label>. All routers have an ERLD of 10 except P1 and P2, which have an ERLD of 4. PE1 is able to push 6 labels, so only a single ELI/EL can be added.

In the example above, adding an ELI/EL after Adj_P1P2 will only allow load-balancing at P1, while inserting it after Adj_PE2P9 will allow load-balancing at P2, P3 ... P9 and maximize the number of LSRs that can perform load-balancing.

7.2.4. Preference for a Part of the Path

An implementation MAY allow the user to favor a part of the end-to-end path when the number of ELI/ELs that can be pushed is not enough to cover the entire path. As an example, a service provider may want to favor load-balancing at the beginning of the path or at the end of the path, so the implementation favors putting the ELI/ELs near the top or the bottom of the stack.

7.2.5. Combining Criteria

An implementation MAY combine multiple criteria to determine the best ELI/ELs placement. However, combining too many criteria could lead to implementation complexity and high resource consumption. Each time the network topology changes, a new evaluation of the ELI/EL placement will be necessary for each impacted LSP.

8. A Simple Example Algorithm

A simple implementation might take into account the ERLD when placing ELI/EL while trying to minimize the number of ELI/ELs inserted and trying to maximize the number of LSRs that can load-balance.

The example algorithm is based on the following considerations:

* An LSR that can insert a limited number of <ELI, EL> pairs should insert such pairs deeper in the stack.

* An LSR should try to insert <ELI, EL> pairs at positions to maximize the number of transit LSRs for which the EL occurs within the ERLD of those LSRs.

* An LSR should try to insert the minimum number of such pairs while trying to satisfy the above criteria.

The pseudocode of the example algorithm is shown below.

```
Initialize the current EL insertion point to the
  bottom-most label in the stack that is EL-capable
while (local-node can push more <ELI,EL> pairs OR
          insertion point is not above label stack) {
    insert an <ELI,EL> pair below current insertion point
    move new insertion point up from current insertion point until
        ((last inserted EL is below the ERLD) AND (ERLD > 2)
                          AND
         (new insertion point is EL-capable))
    set current insertion point to new insertion point
}
```

Figure 8: Example Algorithm to Insert <ELI, EL> Pairs in a Label Stack

When this algorithm is applied to the example described in Section 3, it will result in ELs being inserted in two positions; one after the label L_N-D and another after L_N-P3. Thus, the resulting label stack would be <L_N-P3, ELI, EL, L_A-L1, L_N-D, ELI, EL>.

9. Deployment Considerations

As long as LSR node data-plane capabilities are limited (number of labels that can be pushed or number of labels that can be inspected), hop-by-hop load-balancing of SPRING-encapsulated flows will require trade-offs.

The entropy label is still a good and usable solution as it allows load-balancing without having to perform deep packet inspection on each LSR: It does not seem reasonable to have an LSR inspecting UDP ports within a GRE tunnel carried over a 15-label SPRING tunnel.

Due to the limited capacity of reading a deep stack of MPLS labels, multiple ELI/ELs may be required within the stack, which directly impacts the capacity of the head-end to push a deep stack: each ELI/EL inserted requires two additional labels to be pushed.

Placement strategies of ELI/ELs are required to find the best trade-off. Multiple criteria could be taken into account, and some level of customization (by the user) is required to accommodate different deployments. Since analyzing the path of each destination to determine the best ELI/EL placement may be time consuming for the control plane, we encourage implementations to find the best trade-off between simplicity, resource consumption, and load-balancing efficiency.

In the future, hardware and software capacity may increase data-plane capabilities and may remove some of these limitations, increasing load-balancing capability using entropy labels.

10. Options Considered

Different options that were considered to arrive at the recommended solution are documented in this section.

These options are detailed here only for historical purposes.

10.1. Single EL at the Bottom of the Stack

In this option, a single EL is used for the entire label stack. The source LSR S encodes the entropy label at the bottom of the label stack. In the example described in Section 3, it will result in the label stack at LSR S to look like <L_N-P3, L_A-L1, L_N-D, ELI, EL> <remaining packet header>. Note that the notation in [RFC6790] is used to describe the label stack. An issue with this approach is that as the label stack grows due an increase in the number of SIDs, the EL goes correspondingly deeper in the label stack. Hence, transit LSRs have to access a larger number of bytes in the packet header when making forwarding decisions. In the example described in Section 3, if we consider that the LSR P1 has an ERLD of 3, P1 would

load-balance traffic poorly on the parallel links L3 and L4 since the EL is below the ERLD of P1. A load-balanced network design using this approach must ensure that all intermediate LSRs have the capability to read the maximum label stack depth as required for the application that uses source-routed stacking.

This option was rejected since there exist a number of hardware implementations that have a low maximum readable label depth. Choosing this option can lead to a loss of load-balancing using EL in a significant part of the network when that is a critical requirement in a service-provider network.

10.2. An EL per Segment in the Stack

In this option, each segment/label in the stack can be given its own EL. When load-balancing is required to direct traffic on a segment, the source LSR pushes an <ELI, EL> before pushing the label associated to this segment. In the example described in Section 3, the source label stack that is LSR S encoded would be <L_N-P3, ELI, EL, L_A-L1, L_N-D, ELI, EL>, where all the ELs can be the same. Accessing the EL at an intermediate LSR is independent of the depth of the label stack and hence, independent of the specific application that uses source-routed tunnels with label stacking. A drawback is that the depth of the label stack grows significantly, almost 3 times as the number of labels in the label stack. The network design should ensure that source LSRs have the capability to push such a deep label stack. Also, the bandwidth overhead and potential MTU issues of deep label stacks should be considered in the network design.

This option was rejected due to the existence of hardware implementations that can push a limited number of labels on the label stack. Choosing this option would result in a hardware requirement to push two additional labels per tunnel label. Hence, it would restrict the number of tunnels that can be stacked in an LSP and hence, constrain the types of LSPs that can be created. This was considered unacceptable.

10.3. A Reusable EL for a Stack of Tunnels

In this option, an LSR that terminates a tunnel reuses the EL of the terminated tunnel for the next inner tunnel. It does this by storing the EL from the outer tunnel when that tunnel is terminated and reinserting it below the next inner tunnel label during the label-swap operation. The LSR that stacks tunnels should insert an EL below the outermost tunnel. It should not insert ELs for any inner tunnels. Also, the penultimate hop LSR of a segment must not pop the ELI and EL even though they are exposed as the top labels since the terminating LSR of that segment would reuse the EL for the next segment.

In Section 3, the source label stack that is LSR S encoded would be <L_N-P3, ELI, EL, L_A-L1, L_N-D>. At P1, the outgoing label stack would be <L_N-P3, ELI, EL, L_A-L1, L_N-D> after it has load-balanced to one of the links L3 or L4. At P3, the outgoing label stack would be <L_N-D, ELI, EL>. At P2, the outgoing label stack would be <L_N-D, ELI, EL> and it would load-balance to one of the next-hop LSRs P4 or P5. Accessing the EL at an intermediate LSR (e.g., P1) is independent of the depth of the label stack and hence, independent of the specific use case to which the label stack is applied.

This option was rejected due to the significant change in label-swap operations that would be required for existing hardware.

10.4. EL at Top of Stack

A slight variant of the reusable EL option is to keep the EL at the top of the stack rather than below the tunnel label. In this case, each LSR that is not terminating a segment should continue to keep the received EL at the top of the stack when forwarding the packet along the segment. An LSR that terminates a segment should use the

EL from the terminated segment at the top of the stack when forwarding onto the next segment.

This option was rejected due to the significant change in label swap operations that would be required for existing hardware.

10.5. ELs at Readable Label Stack Depths

In this option, the source LSR inserts ELs for tunnels in the label stack at depths such that each LSR along the path that must load-balance is able to access at least one EL. Note that the source LSR may have to insert multiple ELs in the label stack at different depths for this to work since intermediate LSRs may have differing capabilities in accessing the depth of a label stack. The label stack depth access value of intermediate LSRs must be known to create such a label stack. How this value is determined is outside the scope of this document. This value can be advertised using a protocol such as an IGP.

Applying this method to the example in Section 3, if LSR P1 needs to have the EL within a depth of 4, then the source label stack that is LSR S encoded would be <L_N-P3, ELI, EL, L_A-L1, L_N-D, ELI, EL>, where all the ELs would typically have the same value.

In the case where the ERLD has different values along the path and the LSR that is inserting <ELI, EL> pairs has no limit on how many pairs it can insert, and it knows the appropriate positions in the stack where they should be inserted, this option is the same as the recommended solution in Section 7.

Note that a refinement of this solution, which balances the number of pushed labels against the desired entropy, is the solution described in Section 7.

11. IANA Considerations

This document has no IANA actions.

12. Security Considerations

Compared to [RFC6790], this document introduces the notion of ERLD and MSD, and may require an ingress node to push multiple ELIs/ELs. These changes do not introduce any new security considerations beyond those already listed in [RFC6790].

13. References

13.1. Normative References

[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>.

[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>.

[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>.

[RFC8402] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", RFC 8402, DOI 10.17487/RFC8402, July 2018, <https://www.rfc-editor.org/info/rfc8402>.

[RFC8660] Bashandy, A., Ed., Filsfils, C., Ed., Previdi, S., Litkowski, S., and R. Shakir, "Segment Routing with the MPLS Data Plane", RFC 8660, DOI 10.17487/RFC8660, December

2019, <https://www.rfc-editor.org/info/rfc8660>.

13.2. Informative References

[ISIS-ELC] Xu, X., Kini, S., Psenak, P., Filsfils, C., Litkowski, S., and M. Bocci, "Signaling Entropy Label Capability and Entropy Readable Label Depth Using IS-IS", Work in Progress, Internet-Draft, draft-ietf-isis-mpls-elc-10, 21 October 2019, <https://tools.ietf.org/html/draft-ietf-isis-mpls-elc-10>.

[OSPF-ELC] Xu, X., Kini, S., Psenak, P., Filsfils, C., Litkowski, S., and M. Bocci, "Signaling Entropy Label Capability and Entropy Readable Label-stack Depth Using OSPF", Work in Progress, Internet-Draft, draft-ietf-ospf-mpls-elc-12, 25 October 2019, <https://tools.ietf.org/html/draft-ietf-ospf-mpls-elc-12>.

[RFC8668] Ginsberg, L., Bashandy, A., Filsfils, C., Nanduri, M., and E. Aries, "Advertising Layer 2 Bundle Member Link Attributes in IS-IS", RFC 8668, DOI 10.17487/RFC8668, December 2019, <https://www.rfc-editor.org/info/rfc8668>.

[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>.

[RFC8476] Tantsura, J., Chunduri, U., Aldrin, S., and P. Psenak, "Signaling Maximum SID Depth (MSD) Using OSPF", RFC 8476, DOI 10.17487/RFC8476, December 2018, <https://www.rfc-editor.org/info/rfc8476>.

[RFC8491] Tantsura, J., Chunduri, U., Aldrin, S., and L. Ginsberg, "Signaling Maximum SID Depth (MSD) Using IS-IS", RFC 8491, DOI 10.17487/RFC8491, November 2018, <https://www.rfc-editor.org/info/rfc8491>.

[MSD-BGP] Tantsura, J., Chunduri, U., Talaulikar, K., Mirsky, G., and N. Triantafillis, "Signaling MSD (Maximum SID Depth) using Border Gateway Protocol Link-State", Work in Progress, Internet-Draft, draft-ietf-idr-bgp-ls-segment-routing-msd-09, 15 October 2019, <https://tools.ietf.org/html/draft-ietf-idr-bgp-ls-segment-routing-msd-09>.

Acknowledgements

The authors would like to thank John Drake, Loa Andersson, Curtis Villamizar, Greg Mirsky, Markus Jork, Kamran Raza, Carlos Pignataro, Bruno Decraene, Chris Bowers, Nobo Akiya, Daniele Ceccarelli, and Joe Clarke for their review, comments, and suggestions.

Contributors

Xiaohu Xu
Huawei
Email: xuxiaohu@huawei.com

Wim Hendrickx
Nokia
Email: wim.henderickx@nokia.com

Gunter Van de Velde
Nokia
Email: gunter.van_de_velde@nokia.com

Acee Lindem
Cisco
Email: acee@cisco.com

Authors' Addresses

Sriganesh Kini

Email: sriganeshkini@gmail.com

Kireeti Kompella
Juniper

Email: kireeti@juniper.net

Siva Sivabalan
Cisco

Email: msiva@cisco.com

Stephane Litkowski
Orange

Email: slitkows.ietf@gmail.com

Rob Shakir
Google

Email: robjs@google.com

Jeff Tantsura
Apstra, Inc.

Email: jefftant.ietf@gmail.com