Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 22, 2017

C. Filsfils
S. Sivabalan
Cisco Systems, Inc.
D. Yoyer
Bell Canada.
M. Nanduri
Microsoft Corporation.
S. Lin
A. Bogdanov
Google, Inc.
M. Horneffer
Deutsche Telekom
F. Clad
Cisco Systems, Inc.,
D. Steinberg
Steinberg Consulting
B. Decraene
S. Litkosky
Orange Business Services
February 18, 2017

Segment Routing Policy for Traffic Engineering
draft-filsfils-spring-segment-routing-policy-00.txt

Abstract

Segment Routing (SR) allows a headend node to steer a packet flow along any path. Intermediate per-flow states are eliminated thanks to source routing. The headend node steers a flow into an SR Policy. The header of a packet steered in an SR Policy is augmented with the ordered list of segments associated with that SR Policy. This document details the concepts of SR Policy and steering into an SR Policy.

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 22, 2017.

Table of Contents

1. Introduction

Segment Routing (SR) allows a headend node to steer a packet flow along any path. Intermediate per-flow states are eliminated thanks to source routing [I-D.ietf-spring-segment-routing].

The headend node is said to steer a flow into an Segment Routing Policy (SR Policy).

The header of a packet steered in an SR Policy is augmented with the ordered list of segments associated with that SR Policy.

This document details the concepts of SR Policy and steering into an SR Policy. These apply equally to the MPLS and SRv6 instantiations of segment routing.

For reading simplicity, the illustrations are provided for the MPLS instantiations.

2. SR Traffic Engineering Architecture

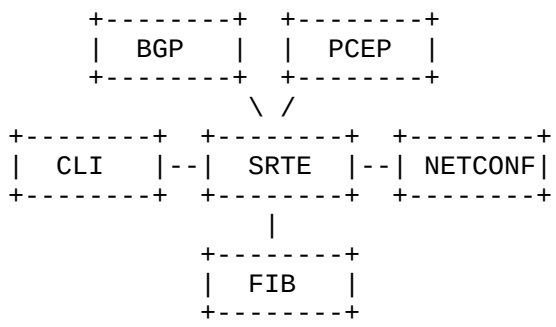

Figure 1: SR Policy architecture

The Segment Routing Traffic Engineering (SRTE) process installs a Segment Routing Policy (SR Policy) in the forwarding plane (FIB).

An SR policy is represented in FIB as a BSID-keyed entry with the action of steering the packets matching this entry to the selected path of the SR Policy.

For a given SR policy, the SRTE process MAY learn multiple candidate paths from different sources: NETCONF with OpenConfig or YANG model (work in progress), PCEP [I-D.ietf-pce-pce-initiated-lsp], local configuration or BGP [I-D.previdi-idr-segment-routing-te-policy].

The SRTE process selects the best candidate path and installs it in FIB.

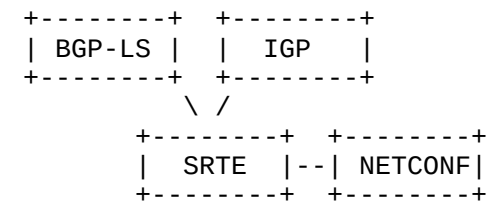

Figure 2: Topology/link-state database architecture

The SRTE process maintains an SRTE database (SRTE-DB).

The SRTE-DB is multi-domain capable.

The attached domain topology MAY be learned via IGP, BGP-LS or NETCONF.

A non-attached (remote) domain topology MAY be learned via BGP-LS or NETCONF.

In some use-cases, the SRTE-DB may only contain the attached domain topology while in others, the SRTE-DB may contain the topology of multiple domains.

3. SR Policy

An SR Policy is identified through the following tuple:

- The head-end where the policy is instantiated/implemented.
- The endpoint (i.e.: the destination of the policy).
- The color (an arbitrary numerical value).

At a given head-end, an SR Policy is fully identified by the <color, endpoint> tuple.

An endpoint can be specified as an IPv4 or IPv6 address.

An SR Policy contain contains one or more candidate paths.

An SR Policy instantiates one single path in RIB/FIB: i.e. the selected path among the candidate paths.

A candidate path is either dynamic or explicit.

A dynamic path expresses an optimization objective and a set of constraints. The headend computes a solution to the optimization problem as a Segment Identifier (SID) list or a set of SID lists. When the headend does not have enough topological information (e.g. multi-domain problem), the headend may delegate the computation to a PCE. Whenever the network situation changes, the path is recomputed.

An explicit path is a SID list or a set of SID lists.

A candidate path has a preference. If not specified, the default preference is 100.

A candidate path is associated with a single Binding SID (BSID).

A candidate path is valid if it is usable. A common path validity criterion is the reachability of its constituent SIDs. The validation rules are defined in a later section.

A Path is selected (i.e. it is the best path of the policy) when it is valid and its preference is the best (highest value) among all the paths of the SR Policy.

Whenever a new path is learned or the validity of an existing path changes or an existing path is changed, the selection process must be re-executed.

A headend may be informed about a path for a policy <color, endpoint> by various means including: local configuration, NETCONF, PCEP or BGP. The protocol source of the path does not matter to the path selection logic.

In the vast majority of use-cases known to date, a path is associated with a single SID list and each path of a policy has a different preference.

The SID list of an SR Policy is the SID list of its selected path.

The BSID of an SR Policy refers to its selected path.

In all the use-cases known to date, all the paths associated with a given policy have the same BSID. One may thus assume that in practice a policy has a stable BSID that is independent of the selected path changes and this BSID is an identification of a policy. However, one should know that a BSID MAY change over the life of an SR Policy and the true identification of a policy is the tuple <headend, endpoint, color>.

An SR Policy <color, endpoint> is active at a headend as soon as this head-end knows about a valid path for this policy.

An active SR Policy installs a BSID-keyed entry in the forwarding plane with the action of steering the packets matching this entry to the SID list of the SR Policy.

If a set of SID lists is associated with the selected path of the policy, then the steering is flow and W-ECMP based according to the relative weight of each SID list.

In summary, the information model is the following:

```
SR policy FOO
    path 200 (selected)
        BSID1
        Weight W1, SID list1: SID11...SID1i
        Weight W2, SID list2: SID21...SID2j
    path 100 (selected)
        BSID2
        Weight W3, SID list3: SID31...SID3i
        Weight W4, SID list4: SID41...SID4j
```

In general BSDIn = BSID1 = BSID2 ...

4. SID List

The segment list (SID list) includes segments of different types (1 to 8) and an optional weight value that is used for W-ECMP.

The following segment types are defined:

Type 1: SID only, in the form of MPLS Label.
Type 2: SID only, in the form of IPv6 address.
Type 3: IPv4 Node Address with optional SID.
Type 4: IPv6 Node Address with optional SID.
Type 5: IPv4 Address + index with optional SID.
Type 6: IPv4 Local and Remote addresses with optional SID.
Type 7: IPv6 Address + index with optional SID.
Type 8: IPv6 Local and Remote addresses with optional SID.

The optional SID can be an MPLS label (SR applied to the MPLS dataplane) or an IPv6 SID (SRv6, SR applied to the IPv6 dataplane).

When building the MPLS label stack or the IPv6 Segment list from the Segment List, the node instantiating the policy MUST interpret the set of Segments as follows:

- The first Segment represents the topmost label or the first IPv6 segment. It identifies the first segment the traffic will be directed toward along the SR explicit path.
- The last Segment represents the bottommost label or the last IPv6 segment the traffic will be directed toward along the SR explicit path.

A SID list is represented as <S1, S2, ... Sn> where S1 is the first SID.

4.1. Explicit Null

A Type 1 SID may be any MPLS label, including reserved labels.

For example, assuming that the desired traffic-engineered path from a headend 1 to an endpoint 4 can be expressed by the SID list <16002, 16003, 16004> where 16002, 16003 and 16004 respectively refer to the IPv4 Prefix SIDs bound to node 2, 3 and 4, then IPv6 traffic can be traffic-engineered from nodes 1 to 4 via the previously described path using an SRTE Policy with SID list <16002, 16003, 16004, 2> where mpls label value of 2 represents the "IPv6 Explicit NULL Label.

The penultimate node before node 4 will pop 16004 and will forward the frame on its directly connected interface to node 4.

The endpoint receives the traffic with top label "2" which indicates that the payload is an IPv6 packet.

5. SR Policy Multi-Domain Database

A headend can learn an attached domain topology via its IGP or a BGP-LS session. A headend can learn a non-attached domain topology via a BGP-LS session.

A headend collects all these topologies in the SR-TE database (SRTE-DB).

The SRTE-DB is multi-domain capable.

In some deployments, the SRTE-DB may only contain the attached domain topology while in others, the SRTE-DB may contain the topology of multiple domains.

6. Operations

6.1. W-ECMP

Packets steered to an SR Policy (i.e. to its BSID either via presence in the packet header as active segment or via FIB recursion) are load-balanced on a weighted basis among the SID lists associated with the selected path of the SR Policy.

The fraction of the flows associated with a given SID list is w/Sw where w is the weight of the SID list and Sw is the sum of the weights of the SID lists of the selected path of the SR Policy.

The accuracy of the weighted load-balancing depends on the platform implementation.

6.2. Path Validation

A SID List is invalid as soon as:

- It is empty.
- The headend is unable to resolve the first SID into one or more outgoing interface(s) and next-hop(s).
- The headend is unable to resolve any non-first SID of type 3-to-8 into an MPLS label or an SRv6 SID.

Unreachable means that the headend has no path to the SID in its SRTE-DB.

In multi-domain deployments, it is expected that the headend be unable to verify the reachability of the SIDs in remote domains. Types 1 and 2 MUST be used for the SIDs for which the reachability cannot be verified. Note that the first SID must always be reachable whatever is type.

A Path is invalid as soon as it has no valid SID list.

The headend of an SR Policy updates the validity of a SID list upon network topological change.

A path of an SR Policy is invalid when all its SID lists are invalid.

An SR Policy is invalid when all its paths are invalid.

6.3. Fast Convergence

Upon topological change, many policies could be recomputed. An implementation MAY provide a per-policy priority field. The operator MAY set this field to indicate in which order the policies should be re-computed. Such a priority may be represented by an integer in the range [0, 254] where the lowest value is the highest priority.

7. Binding SID

7.1. Benefits

The Binding SID (BSID) is fundamental to Segment Routing. It provides scaling, network opacity and service independence.

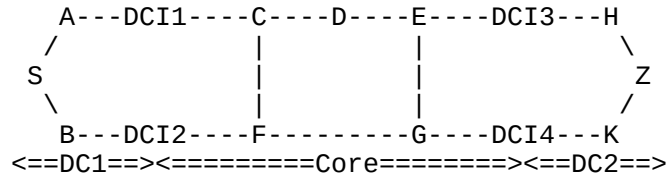

Figure 3: A Simple Datacenter Topology

A simplified illustration is provided on the basis of the previous diagram where we assume that S, A, B, Data Center Interconnect DCI1 and DCI2 share the same IGP-SR instance in the data-center 1 (DC1). DCI1, DCI2, C, D, E, F, G, DCI3 and DCI4 share the same IGP-SR domain

in the core. DCI3, DCI4, H, K and Z share the same IGP-SR domain in the data-center 2 (DC2).

In this example, we assume no redistribution between the IGP's and no presence of BGP. The inter-domain communication is only provided by SR through SR Policies.

The latency from S to DCI1 equals to DCI2. The latency from Z to DCI3 equals to DCI4. All the intra-DC links have the same IGP metric 10.

The path DCI1, C, D, E, DCI3 has a lower latency and lower capacity than the path DCI2, F, G, DCI4.

The IGP metrics of all the core links are set to 10 except the links D-E which is set to 100.

A low-latency multi-domain policy from S to Z may be expressed as <DCI1, BSID, Z> where:

- DCI1 is the prefix SID of DCI1.
- BSID is the Binding SID bound to an SRTE policy <D, D2E, DCI3> instantiated at DCI1.
- Z is the prefix SID of Z.

Without the use of an intermediate core SR Policy (efficiently summarized by a single BSID), S would need to steer its low-latency flow into the policy <DCI1, D, D2E, DCI3, Z>.

The use of a BSID (and the intermediate bound SR Policy) decreases the number of segments imposed by the source.

A BSID acts as a stable anchor point which isolates one domain from the churn of another domain. Upon topology changes within the core of the network, the low-latency path from DCI1 to DCI3 may change. While the path of an intermediate policy changes, its BSID does not change. Hence the policy used by the source does not change, hence the source is shielded from the churn in another domain.

A BSID provides opacity and independence between domains. The administrative authority of the core domain may not want to share information about its topology. The use of a BSID allows keeping the service opaque. S is not aware of the details of how the low-latency service is provided by the core domain. S is not aware of the need of the core authority to temporarily change the intermediate path.

7.2. Allocation

There are three approaches to allocate a BSID to an SR Policy: all the paths have no explicit BSID (called dynamic allocation), all the paths have the same explicit BSID (explicit allocation) and finally a mix of paths with and without explicit BSID (generic allocation).

In practice, all the use-cases seen to-date either use the explicit allocation or the dynamic allocation. The explicit allocation is most-often associated with controller-instantiated SR Policies. The dynamic allocation is most-often associated with router-based on-demand SR Policies.

7.2.1. Dynamic BSID Allocation

No path of the SR Policy have a specified BSID.

In such a case, the SR-TE implementation allocates a SID to the SR Policy and keeps it along the whole existence of the policy.

In the case of SR-MPLS, the SR-TE implementation binds a local dynamic label in the same way LDP, RSVP-TE or BGP would do.

7.2.2. Explicit BSID Allocation

All the paths of the SR Policy have the same specified BSID, with the same behavioral preference in case this specified BSID is not available.

If the specified BSID is available, then it is bound to the SR Policy and used along the existence of the policy.

If the specified BSID is not available, then a SYSLOG/NETCONF message is generated and if the preferred behavior is to fall-back on the dynamic allocation, then the dynamic allocation is performed.

If the specified BSID is not available and the operator-requested behavior is to not fall-back on the dynamic allocation, then a SYSLOG/NETCONF message is generated and the SR Policy does not install any BSID entry in the forwarding plane.

A later section will explain how controllers can discover the local SIDs available at a node N so as to pick an explicit BSID for a SR Policy to be instantiated at headend N.

7.2.3. Generic BSID Allocation

This section details the BSID allocation when a policy is made of paths with different BSID allocation behaviors (e.g. mix of paths with and without an explicit BSID, potentially with different explicit BSIDs).

When the selected path has a specified BSID, the SR Policy uses that BSID if this value (label in MPLS, IPv6 address in SRv6) is available (i.e. not associated with any other usage: e.g. to another MPLS client, to another SID, to another SR Policy).

If the selected path's BSID is not available, then the SR Policy keeps the previous BSID. If the SR Policy did not have a previous BSID, then the SR Policy dynamically binds a BSID to itself.

Note that a path may request that only its specified BSID be used. In that case, if that BSID is not available and that path is active, then no BSID is bound to the policy and a SYSLOG/NETCONF is triggered. In this case, the SR Policy does not install any entry indexed by a BSID in the forwarding plane.

When an SR Policy has multiple multiple valid paths with the best preference but with different BSIDs, it is left to the implementation to decide which BSID to install. This case is unlikely in practice for two reasons. First, all known use-cases share the same BSID across all the paths of a given SR Policy. Second, all known use-cases have a different preference for each path. Hence in practice a single path will be active and with a stable BSID on a per-policy basis.

8. Centralized Discovery

This section explains how controllers can discover the local SIDs available at a node N so as to pick an explicit BSID for a SR Policy to be instantiated at headend N.

Any controller can discover the following properties of a node N (e.g. via BGP-LS, NETCONF etc.):

- its local Segment Routing Label Block (SRLB).
- its local topology.
- its topology-related SIDs (Adj SID and EPE SID).
- its SR Policies and their BSID ([I-D.ietf-idr-te-lsp-distribution]).

Any controller can thus infer the available SIDs in the SRLB of any node.

As an example, a controller discovers the following characteristics of N: SRLB [4000, 8000], 3 Adj SIDs (4001, 4002, 4003), 2 EPE SIDs (4004, 4005) and 3 SRTE policies (whose BSIDs are respectively 4006, 4007 and 4008). This controller can deduce that the SRLB sub-range [4009, 5000] is free for allocation.

Likely, the next question is: how do we ensure that different controllers do not pick the same available SID at the same time for different SR Policies.

Clearly, a controller is not restricted to use the next numerically available SID in the available SRLB sub-range. It can pick any label in the subset of available labels. This random pick make the chance for a collision unlikely.

An operator could also sub-allocate the SRLB between different controllers (e.g. [4000-4499] to controller 1 and [4500-5000] to controller 2).

Inter-controller state-synchronization may be used to avoid/detect collision in BSID.

All these techniques make the likelihood of a collision between different controllers very unlikely.

In the unlikely case of a collision, the controllers will detect it through SYSLOG/NETCONF, BGP-LS reporting ([I-D.ietf-idr-te-lsp-distribution]) or PCEP notification. They then have the choice to continue the operation of their SR Policy with the dynamically allocated BSID or re-try with another explicit pick.

Note: in deployments where PCE Protocol (PCEP) is used between head-end and controller (PCE), a head-end can report BSID as well as policy attributes (e.g., type of disjointness) and operational and administrative states to controller. Similarly, a controller can also assign/update the BSID of a policy via PCEP when instantiating or updating SR Policy.

9. Dynamic Path

A dynamic path is a path that expresses an optimization objective and constraints.

The headend of the policy is responsible to compute a SID list ("solution SID list") that fits this optimization problem. The headend is responsible for computing the solution SID list any time the inputs to the problem change (e.g. topology changes).

9.1. Optimization Objective

We define two optimization objectives:

- Min-Metric - requests computation of a solution SID list optimized for a selected metric.
- Min-Metric with margin and maximum number of SIDs - Min-Metric with two changes: a margin of by which two paths with similar metrics would be considered equal, a constraint on the max number of SIDs in the SID list.

The "Min-Metric" optimization objective requests to compute a solution SID list such that packets flowing through the solution SID list use ECMP-aware paths optimized for the selected metric. The "Min-Metric" objective can be instantiated for the IGP metric xor the TE metric xor the latency extended TE metric. This metric is called the O metric (the optimized metric) to distinguish it from the IGP metric. The solution SID list must be computed to minimize the number of SIDs and the number of SID lists.

If the selected O metric is the IGP metric and the headend and tailend are in the same IGP domain, then the solution SID list is made of the single prefix-SID of the tailend.

When the selected O metric is not the IGP metric, then the solution SID list is made of prefix SIDs of intermediate nodes, Adjacency SIDs along intermediate links and potentially BSIDs of intermediate policies.

In many deployments there are insignificant metric differences between mostly equal path (e.g. a difference of 100 usec of latency between two paths from NYC to SFO would not matter in most cases). The "Min-Metric with margin" objective supports such requirement.

The "Min-Metric with margin and maximum number of SIDs" optimization objective requests to compute a solution SID list such that packets flowing through the solution SID list do not use a path whose cumulated O metric is larger than the shortest-path O metric + margin.

If this is not possible because of the number of SIDs constraint, then the solution SID list minimizes the O metric while meeting the maximum number of SID constraints.

9.2. Constraints

The following constraints can be defined:

- Inclusion and/or exclusion of TE affinity.
- Inclusion and/or exclusion of IP address.
- Inclusion and/or exclusion of SRLG.
- Inclusion and/or exclusion of admin-tag.
- Maximum accumulated metric (IGP, TE and latency).
- Maximum number of SIDs in the solution SID list.
- Maximum number of weighted SID lists in the solution set.
- Diversity to another service instance (e.g., link, node, or SRLG disjoint paths originating from different head-ends).

9.3. SR Native Algorithm

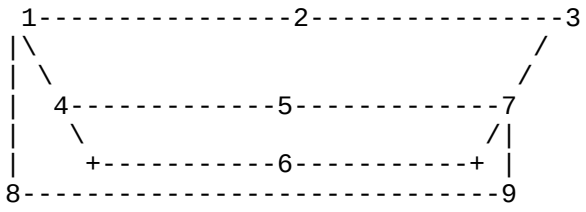

Figure 4: Illustration used to describe SR native algorithm

Let us assume that all the links have the same IGP metric of 10 and let us consider the dynamic path defined as: Min-Metric(from 1, to 3, IGP metric, margin 0) with constraint "avoid link 2-to-3".

A classical circuit implementation would do: prune the graph, compute the shortest-path, pick a single non-ECMP branch of the ECMP-aware shortest-path and encode it as a SID list. The solution SID list would be <4, 5, 7, 3>.

An SR-native algorithm would find a SID list that minimizes the number of SIDs and maximize the use of all the ECMP branches along the ECMP shortest path. In this illustration, the solution SID list would be <7, 3>.

In the vast majority of SR use-cases, SR-native algorithms should be preferred: they preserve the native ECMP of IP and they minimize the dataplane header overhead.

In some specific use-case (e.g. TDM migration over IP where the circuit notion prevails), one may prefer a classic circuit computation followed by an encoding into SIDs.

SR-native algorithms are a local node behavior and are thus outside the scope of this document.

9.4. Path to SID

Let us assume the below diagram where all the links have an IGP metric of 10 and a TE metric of 10 except the link AB which has an IGP metric of 20 and the link AD which has a TE metric of 100. Let us consider the min-metric(from A, to D, TE metric, margin 0).

Figure 5: Illustration used to describe path to SID conversion

The solution path to this problem is ABCD.

This path can be expressed in SIDs as $#60;B, D$#62; where B and D are the IGP prefix SIDs respectively associated with nodes B and D in the diagram.

Indeed, from A, the IGP path to B is AB (IGP metric 20 better than ADCB of IGP metric 30). From B, the IGP path to D is BCD (IGP metric 20 better than BAD of IGP metric 30).

While the details of the algorithm remain a local node behavior, a high-level description follows: start at the headend and find an IGP prefix SID that leads as far down the desired path as possible (without using any link not included in the desired path). If no prefix SID exists, use the Adj SID to the first neighbor along the path. Restart from the node that was reached.

9.5. PCE Computed Path

A local computation should be preferred whenever possible. When local computation is not possible (e.g., a policy’s tail-end is outside the topology known to the head-end), the head-end may send path computation request to a PCE supporting PCEP extension specified in [I-D.ietf-pce-segment-routing].

10. Signaling Paths of an SR Policy to a Head-end

A headend H can be informed about a path for an SR policy (endpoint, color) via several means: BGP, PCEP, CLI, netconf.

We remind that the selection of the best path for a policy is independent of the protocol source of the path.

10.1. BGP

Please refer to [I-D.previdi-idr-segment-routing-te-policy]

10.2. PCEP

Please refer to [I-D.ietf-pce-pce-initiated-lsp]

10.3. NETCONF

Operator MUST be able to install policy via NETCONF with OpenConfig/ YANG models (work in progress).

10.4. CLI

Operator MUST be able to install policy via CLI.

11. Steering into an SR Policy

A headend can steer a packet flow on an SR Policy in various ways:

- Incoming packets have an active SID matching a local BSID at the head-end.
- Incoming packets match a BGP/Service route which recurses on the BSID of a local policy.
- Incoming packets match a BGP/Service route which recurses on an array of paths to the BGP nhop where some of the paths in the array are local SR Policies.
- Incoming packets match a routing policy which directs them on a local SR policy.

For simplicity of illustration, we will use the SR-MPLS example.

11.1. Incoming Active SID is a BSID

Let us assume that headend H has a local SR Policy P of SID list <S1, S2, S3> and BSID B.

When H receives a packet with label stack <B, L2, L3>, H pops B and pushes <S1, S2, S3>. H sends the resulting packet with label stack <S1, S2, S3, L2, L3> along the path to S1.

H has steered the packet in the policy P.

H did not have to classify the packet. The classification was done by a node upstream of H (e.g. the source of the packet or an intermediate ingress edge node of the SR domain) and the result of this classification was efficiently encoded in the packet header as a BSID.

This is another key benefit of the segment routing in general and the binding SID in particular: the ability to encode a classification and the resulting steering in the packet header such as to better scale and simplify intermediate aggregation nodes.

11.2. Recursion on a BSID

Let us assume that headend H:

- learns about a BGP route R/r via next-hop N, extended-color community C and label V.
- has a local SR Policy P to (endpoint = N, color = C) of SID list <S1, S2, S3> and BSID B.
- has a local BGP policy which matches on the extended-color community C and allows its usage as an SR-TE SLA steering information.

In such a case, H installs R/r in RIB/FIB with next-hop = B (instead of N).

Indeed, H’s local BGP policy and the received BGP route indicate that the headend should associate R/r with an SR-TE path to N with the SLA associated with color C. The headend therefore installs the BGP route on that policy.

This can be implemented by using the BSID as a generalized nhop and installing the BGP route on that generalized next-hop.

When H receives a packet with a destination matching R/r, H pushes the label stack <S1, S2, S3, V> and sends the resulting packet along the path to S1.

Note that any label associated with the BGP route is pushed after the SID list of the SR Policy.

11.3. Recursion on a dynamic BSID

In the previous section, we assumed that H had a pre-established "explicit" SR Policy (endpoint N, color C).

In this section, we note that this policy may be generated dynamically by the head-end H upon reception of the BGP route R/r via N with color C.

A possible implementation has the BGP policy matches on the color C and triggers an on-demand local request to the SR-TE process to instantiate an SR Policy (endpoint N, color C). Color C is bound to some optimization objective and constraints specified in the local BGP policy defined for color C. Once the related SR Policy is instantiated, the SR-TE process returns the related BSID to BGP process which can then installs the BGP route R/r on B.

The rest of the explanation is the same as the previous section.

11.4. An array of BSIDs associated with an IGP entry

Let us assume that head-end H:

- learns about a BGP route R/r via next-hop N and label V.
- has a local SR Policy P1 to (endpoint = N, color = C1) of SID list <S1, S2, S3> and BSID B1.
- has a local SR Policy P2 to (endpoint = N, color = C2) of SID list <S4, S5, S6> and BSID B2.
- is configured to instantiate an array of paths to N where the entry 0 is the IGP path to N, color C1 is the first entry and Color C2 is the second entry. The index into the array is called a Forwarding Class (FC). The index can have values 0 to 7.
- is configured to match flows in its ingress interfaces (upon any field such as Ethernet destination/source/vlan/tos or IP destination/source/DSCP or transport ports etc.) and color them with an internal per-packet forwarding-class variable (0, 1 or 2 in this example).

In such a case, H installs in RIB/FIB:

- R/r in with next-hop N (as usual).
- N via a recursion on an array A (instead of the immediate outgoing link associated with the IGP shortest-path to N.
- Entry A(0) set to the immediate outgoing link of the IGP shortest-path to N.
- Entry A(1) set to B1.
- Entry A(2) set to B2.

H receives three packets P, P1 and P2 on its incoming interface. H colors them respectively with forwarding-class 0, 1 and 2. As a result:

- H pushes <V> on packet P and forwards the resulting frame along the shortest-path to N (which in SR-MPLS results in the pushing of the prefix-SID of N.
- H pushes <S1, S2, S3, V> on packet P1 and forwards the resulting frame along the shortest-path to S1.
- H pushes <S4, S5, S6, V> on packet P2 and forwards the resulting frame along the shortest-path to S4.

If the local configuration does not specify any explicit forwarding information for an entry of the array, then this entry is filled with the same information as entry 0 (i.e. the IGP shortest-path).

This realizes per-flow steering: different flows bound to the same BGP destination R/r are steered on different SR-TE paths.

11.5. A Routing Policy on a BSID

Finally, headend H may be configured with a local routing policy which overrides any BGP/IGP path and steer a specified flow on an SR Policy.

12. Optional Steering Modes for BGP Destinations

12.1. Color-Only BGP Destination Steering

In the previous section "Recursion on a BSID", we have seen that the steering on an SR Policy is governed by the matching of the BGP route’s next-hop N and the authorized color C with a local SR Policy defined by the tuple (N, C).

This is the most likely form of BGP destination steering and the one we recommend.

In this section, we define an alternative steering mechanism based only on the color.

This color-only steering variation is governed by two new flags "C" and "O" defined in the color extended community.

The Color-Only flags "CO" are set to 00 by default.

When 00, the BGP destination is preferably steered onto a valid SR Policy (N, C) where N is an IPv4/6 endpoint address and C is a color

value else it is steered on the IGP path to the next-hop N. This is the classic case we described before and that we recommend.

When 01, the BGP destination is preferably steered onto a valid SR Policy (N, C) else onto a valid SR Policy (null endpoint, C) else on the IGP path to the next-hop N.

When 10, the BGP destination is preferably steered onto a valid SR Policy (N, C) else onto a valid SR Policy (null endpoint, C) else on any valid SR Policy (any endpoint, C) else on the IGP path to the next-hop N.

The null endpoint is 0.0.0.0 for IPv4 and ::0 for IPv6 (all bits set to the 0 value).

When 11, it is treated like 00.

12.2. Drop on Invalid

The local BGP policy authorizing the use of an extended color community steering on an SR policy may specify that if the related SR Policy becomes invalid then the related BSID should remain in RIB/FIB and point to null0 (drop any packet recursing on that BSID).

Recall that, by default, for a BGP route R/r via next-hop N with extended-color community C, when the SR Policy (N, C) becomes invalid, then BGP re-installs R/r in RIB/FIB via N (the IGP path to N).

13. Multipoint SR Policy

13.1. Spray SR Policy

A Spray SR-TE policy is a variant of an SR-TE policy which involves packet replication.

Any traffic steered into a Spray SR Policy is replicated along the SID lists of its selected path.

In the context of a Spray SR Policy, the selected path SHOULD have more than one SID list. The weights of the SID lists is not applicable for a Spray SR Policy. They MUST be set to 1.

Like any SR policy, a Spray SR Policy has a BSID instantiated into the forwarding plane.

Traffic is typically steered into a Spray SR Policy in two ways:

- o local policy-based routing at the headend of the policy.
- o remote classification and steering via the BSID of the Spray SR Policy.

14. Reporting SR Policy

Stateful PCEP ([I-D.ietf-pce-stateful-pce] and [I-D.sivabalan-pce-binding-label-sid] provides an ability for head-end to report BSID, attributes, and operational/administrative states. Using this protocol, a PCE can also update an existing SR Policy whose path computation is delegated to it as well as instantiate new SR Policy on a head-end.

BGP-LS reports an SR Policy via ([I-D.ietf-idr-te-lsp-distribution]

15. Work in Progress

- o Open configuration model.
- o Yang model.

16. Acknowledgement

17. Normative References

[GLOBECOM]
Filsfils, C., Nainar, N., Pignataro, C., Cardona, J., and P. Francois, "The Segment Routing Architecture, IEEE Global Communications Conference (GLOBECOM)", 2015.

[I-D.ietf-idr-te-lsp-distribution]
Previdi, S., Dong, J., Chen, M., Gredler, H., and j. jefftant@gmail.com, "Distribution of Traffic Engineering (TE) Policies and State using BGP-LS", draft-ietf-idr-te-lsp-distribution-06 (work in progress), January 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-09 (work in progress), October 2016.

[I-D.ietf-pce-pce-initiated-lsp]
Crabbe, E., Minei, I., Sivabalan, S., and R. Varga, "PCEP Extensions for PCE-initiated LSP Setup in a Stateful PCE Model", draft-ietf-pce-pce-initiated-lsp-07 (work in progress), July 2016.

[I-D.ietf-pce-segment-routing]
Sivabalan, S., Medved, J., Filsfils, C., Crabbe, E., Raszuk, R., Lopez, V., Tantsura, J., Henderickx, W., and J. Hardwick, "PCEP Extensions for Segment Routing", draft-ietf-pce-segment-routing-08 (work in progress), October 2016.

[I-D.ietf-pce-stateful-pce]
Crabbe, E., Minei, I., Medved, J., and R. Varga, "PCEP Extensions for Stateful PCE", draft-ietf-pce-stateful-pce-18 (work in progress), December 2016.

[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-11 (work in progress), February 2017.

[I-D.previdi-idr-segment-routing-te-policy]
Previdi, S., Filsfils, C., Sreekantiah, A., Sivabalan, S., Mattes, P., Rosen, E., and S. Lin, "Advertising Segment Routing Traffic Engineering Policies in BGP", draft-previdi-idr-segment-routing-te-policy-03 (work in progress), December 2016.

[I-D.sivabalan-pce-binding-label-sid]
Sivabalan, S., Filsfils, C., Previdi, S., Tantsura, J., Hardwick, J., and M. Nanduri, "Carrying Binding Label/ Segment-ID in PCE-based Networks.", draft-sivabalan-pce-binding-label-sid-02 (work in progress), October 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>.

[SIGCOMM] Hartert, R., Vissicchio, S., Schaus, P., Bonaventure, O., Filsfils, C., Telkamp, T., and P. Francois, "A Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks, ACM SIGCOMM", 2015.

Authors' Addresses

Clarence Filsfils
Cisco Systems, Inc.
Pegasus Parc
De kleetlaan 6a, DIEGEM BRABANT 1831
BELGIUM

Email: cfilsfil@cisco.com

Siva Sivabalan
Cisco Systems, Inc.
2000 Innovation Drive
Kanata, Ontario K2K 3E8
Canada

Email: msiva@cisco.com

Daniel Yoyer
Bell Canada.

Email: daniel.yoyer@bell.ca

Mohan Nanduri
Microsoft Corporation.
One Microsoft Way
Redmond, WA 98052
USA

Email: mnanduri@microsoft.com

Steven Lin
Google, Inc.

Email: stevenlin@google.com

Alex Bogdanov
Google, Inc.

Email: bogdanov@google.com

Martin Horneffer
Deutsche Telekom

Email: martin.horneffer@telekom.de

Francois Clad
Cisco Systems, Inc.,

Email: fclad@cisco.com

Dirk Steinberg
Steinberg Consulting

Email: dws@steinbergnet.net

Bruno Decraene
Orange Business Services

Email: bruno.decraene@orange.com

Stephane Litkosky
Orange Business Services

Email: stephane.litkowski@orange.com

SPRING
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2017

C. Fisfils
Cisco Systems, Inc.
J. Leddy
Comcast
D. Voyer
D. Bernier
Bell Canada
D. Steinberg
Steinberg Consulting
R. Raszuk
Bloomberg LP
S. Matsushima
SoftBank Telecom
D. Lebrun
Universite catholique de Louvain
B. Decraene
Orange
B. Peirens
Proximus
S. Salsano
Universita di Roma "Tor Vergata"
G. Naik
Drexel University
H. Elmalky
Ericsson
P. Jonnalagadda
M. Sharif
Barefoot Networks
A. Ayyangar
Arista
S. Mynam
Dell Force10 Networks
A. Bashandy
K. Raza
D. Dukes
F. Clad
P. Camarillo, Ed.
Cisco Systems, Inc.
March 9, 2017

SRv6 Network Programming
draft-filsfils-spring-srv6-network-programming-00

Abstract

This document describes the SRv6 network programming concept and its most basic functions.

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2017.

Table of Contents

1. Introduction

This document defines the SRv6 network programming concept, its most frequent functions and the related terminology.

This document assumes familiarity with the Segment Routing Header [I-D.ietf-6man-segment-routing-header].

2. Terminology

SRH is the abbreviation for the Segment Routing Header. We assume that the SRH may be present multiple times inside each packet.

NH is the abbreviation of the IPv6 next-header field.

NH=SRH means that the next-header field is 43 with routing type 4.

When there are multiple SRHs, they must follow each other: the next-header field of all SRH except the last one must be SRH.

The effective next-header (ENH) is the next-header field of the IP header when no SRH is present, or is the next-header field of the last SRH.

In this version of the document, we assume that there is no other extension header than the SRH. These will be lifted in future versions of the document.

SID: A Segment Identifier which represents a specific segment in segment routing domain. The SID type used in this document is IPv6 address (also referenced as SRv6 Segment or SRv6 SID).

A SID list is represented as <S1, S2, S3> where S1 is the first SID to visit, S2 is the second SID to visit and S3 is the last SID to visit along the SR path.

(SA,DA) (S3, S2, S1; SL) represents an IPv6 packet with:

- IPv6 header with source and destination addresses respectively SA and DA and next-header is SRH

- SRH with SID list <S1, S2, S3> with SegmentsLeft = SL

- Note the difference between the <> and () symbols: <S1, S2, S3> represents a SID list where S1 is the first SID and S3 is the last SID. (S3, S2, S1; SL) represents the same SID list but encoded in the SRH format where the rightmost SID in the SRH is the first SID and the leftmost SID in the SRH is the last SID. When referring to an SR policy in a high-level use-case, it is simpler to use the <S1, S2, S3> notation. When referring to an illustration of the detailed behavior, the (S3, S2, S1; SL) is more convenient.

- The payload of the packet is omitted.

SRH[SL] represents the SID pointed by the SL field in the first SRH. In our example, SRH[2] represents S1, SRH[1] represents S2 and SRH[0] represents S3.

FIB is the abbreviation for the forwarding table. A FIB lookup is a lookup in the forwarding table. When a packet is intercepted on a wire, it is possible that SRH[SL] is different from the DA.

3. SRv6 Segment

An SRv6 Segment is a 128-bit value. "SID" (abbreviation for Segment Identifier) is often used as a shorter reference for "SRv6 Segment".

An SRv6-capable node N maintains a "My Local SID Table". This table contains all the local SRv6 segments explicitly instantiated at node N. N is the parent node for these SIDs.

A local SID of N can be an IPv6 address associated to a local interface of N but it is not mandatory. Nor is the My Local SID table populated by default with all IPv6 addresses defined on node N.

In most use-cases, a local SID will NOT be an address associated to a local interface of N.

A local SID of N could be routed to N but it does not have to be. Most often, it is routed to N via a shorter-mask prefix.

Let’s provide a classic illustration.

Node N is configured with a loopback0 interface address of C1::1/40 originated in its IGP. Node N is configured with two SIDs: C1::100 and C2::101.

The entry C1::1 is not defined explicitly as an SRv6 SID and hence does not appear in the "My Local SID Table". The entries C1::100 and C2::101 are defined explicitly as SRv6 SIDs and hence appear in the "My Local SID Table".

The network learns about a path to C1::/40 via the IGP and hence a packet destined to C1::100 would be routed up to N. The network does not learn about a path to C2::/40 via the IGP and hence a packet destined to C2::101 would not be routed up to N.

A packet could be steered to a non-routed SID C2::101 by using a SID list <...,C1::100,C2::101,...> where the non-routed SID is preceded by a routed SID to the same node. This is similar to the local vs global segments in SR-MPLS.

Every SRv6 local SID instantiated has a specific instruction bound to it. This information is stored in the "My Local SID Table". The "My Local SID Table" has three main purposes:

- Define which local SIDs are explicitly instantiated

- Specify which instruction is bound to each of the instantiated SIDs

- Store the parameters associated with such instruction (i.e. OIF, NextHop,...)

We represent an SRv6 local SID as LOC:FUNCT where LOC is the L most significant bits and FUNCT is the 128-L least significant bits. L is called the locator length and is flexible. Each operator is free to

use the locator length it chooses. Most often the LOC part of the SID is routable and leads to the node which owns that SID.

Often, for simplicity of illustration, we will use a locator length of 64 bits. This is just an example. Implementations must not assume any a priori prefix length.

The FUNCT part of the SID is an opaque identification of a local function bound to the SID. Hence the name SRv6 Local SID.

A function may require additional arguments that would be placed in the rightmost-bits of the 128-bit space. In such case, the SRv6 Local SID will have the form LOC:FUNCT:ARGS.

These arguments may vary on a per-packet basis and may contain information related to the flow, service, or any other information required by the function associated to the SRv6 Local SID.

For to this reason, the "My Local SID Table" matches on a per longest-prefix-match basis.

A node may receive a packet with an SRv6 SID in the DA without an SRH. In such case the packet should still be processed by the Segment Routing engine.

4. Functions associated with a Local SID

Each entry of the "My Local SID Table" indicates the function associated with the local SID.

We define hereafter a set of well-known functions that can be associated with a SID.

End	Endpoint function The SRv6 instantiation of a prefix SID
End.X	Endpoint function with Layer-3 cross-connect The SRv6 instantiation of a Adj SID
End.T	Endpoint function with specific IPv6 table lookup
End.DX2	Endpoint with decapsulation and Layer-2 cross-connect L2VPN use-case
End.DX6	Endpoint with decapsulation and IPv6 cross-connect IPv6 L3VPN use (equivalent of a per-CE VPN label)
End.DX4	Endpoint with decapsulation and IPv4 cross-connect IPv4 L3VPN use (equivalent of a per-CE VPN label)
End.DT6	Endpoint with decapsulation and IPv6 table lookup IPv6 L3VPN use (equivalent of a per-VRF VPN label)
End.DT4	Endpoint with decapsulation and IPv4 table lookup IPv4 L3VPN use (equivalent of a per-VRF VPN label)
End.B6	Endpoint bound to an SRv6 policy SRv6 instantiation of a Binding SID
End.B6.Encaps	Endpoint bound to an SRv6 encapsulation Policy SRv6 instantiation of a Binding SID
End.BM	Endpoint bound to an SR-MPLS Policy SRv6/SR-MPLS instantiation of a Binding SID
End.S	Endpoint in search of a target in table T
End.AS	Endpoint to SR-unaware APP via static proxy
End.AM	Endpoint to SR-unaware APP via masquerading

The list is not exhaustive. In practice, any function can be attached to a local SID: e.g. a node N can bind a SID to a local VM or container which can apply any complex function on the packet.

We call N the node who has an explicitly defined local SID S and we detail the function that N binds to S.

At the end of this section we also present some flavours of these well-known functions.

4.1. End: Endpoint

The Endpoint function ("End" for short) is the most basic function.

When N receives a packet whose IPv6 DA is S and S is a local End SID, N does:

```
1.   IF NH=SRH and SL > 0
2.      decrement SL
3.      update the IPv6 DA with SRH[SL]
4.      FIB lookup on updated DA                          ;; Ref1
5.      forward accordingly to the matched entry          ;; Ref2
6.   ELSE
7.      drop the packet                                   ;; Ref3
```

Ref1: The End function performs the FIB lookup in the forwarding table associated to the ingress interface

Ref2: If the FIB lookup matches a multicast state, then the related RPF check must be considered successful

Ref3: a local SID could be bound to a function which authorizes the decapsulation of an outer header (e.g. IPinIP) or the punting of the packet to TCP, UDP or any other protocol. This however needs to be explicitly defined in the function bound to the local SID. By default, a local SID bound to the well-known function "End" only allows the punting to OAM protocols and neither allows the decapsulation of an outer header nor the cleanup of an SRH. As a consequence, an End SID cannot be the last SID of an SRH and cannot be the DA of a packet without SRH.

This is the SRv6 instantiation of a Prefix SID [I-D.ietf-spring-segment-routing].

4.2. End.X: Endpoint with Layer-3 cross-connect

The "Endpoint with cross-connect to an array of layer-3 adjacencies" function (End.X for short) is a variant of the End function.

When N receives a packet destined to S and S is a local End.X SID, N does:

```
1.   IF NH = SRH and SL > 0
2.      decrement SL
3.      update the IPv6 DA with SRH[SL]
4.      forward to layer-3 adjacency bound to the SID S   ;; Ref1
5.   ELSE
6.      drop the packet                                   ;; Ref2
```

Ref1: If an array of adjacencies is bound to the End.X SID, then one entry of the array is selected based on a hash of the packet's header.

Ref2: An End.X function only allows punting to OAM and does not allow decaps. An End.X SID cannot be the last SID of an SRH and cannot be the DA of a packet without SRH.

The End.X function is required to express any traffic-engineering policy.

This is the SRv6 instantiation of an Adjacency SID [I-D.ietf-spring-segment-routing].

If a node N has 30 outgoing interfaces to 30 neighbors, usually the operator would explicitly instantiate 30 End.X SIDs at N: one per layer-3 adjacency to a neighbor. Potentially, more End.X could be explicitly defined (groups of layer-3 adjacencies to the same neighbor or to different neighbors).

Note that with SR-MPLS, an AdjSID is typically preceded by a PrefixSID. This is unlikely in SRv6 as most likely an End.X SID is globally routed to N.

Note that if N has an outgoing interface bundle I to a neighbor Q made of 10 member links, N may allocate up to 11 End.X local SIDs for that bundle: one for the bundle itself and then up to one for each member link. This is the equivalent of the L2-Link Adj SID in SR-MPLS [I-D.ietf-isis-l2bundles].

4.3. End.T: Endpoint with specific IPv6 table lookup

The "Endpoint with specific IPv6 table lookup" function (End.T for short) is a variant of the End function.

When N receives a packet destined to S and S is a local End.T SID, N does:

```
1.   IF NH=SRH and SL > 0                                    ;; Ref1
2.      lookup the next segment in IPv6 table T associated with the SID
3.      forward via the matched table entry
4.   ELSE
5.      drop the packet
```

Ref1: The End.T SID must not be the last SID

The End.T is used for multi-table operation in the core.

4.4. End.DX2: Endpoint with decapsulation and Layer-2 cross-connect

The "Endpoint with decapsulation and Layer-2 cross-connect to OIF" function (End.DX2 for short) is a variant of the endpoint function.

When N receives a packet destined to S and S is a local End.DX2 SID, N does:

```
1.   IF NH=SRH and SL > 0
2.      drop the packet                                        ;; Ref1
3.   ELSE IF ENH  = 59                                        ;; Ref2
4.      pop the (outer) IPv6 header and its extension headers
5.      forward the resulting frame via OIF associated to the SID
6.   ELSE
7.      drop the packet
```

Ref1: An End.DX2 SID must always be the last SID, or it can be the Destination Address of an IPv6 packet with no SRH header.

Ref2: We conveniently reuse the next-header value 59 allocated to IPv6 No Next Header [RFC2460]. When the SID is of function End.DX2 and the Next-Header=59, we know that an Ethernet frame is in the payload without any further header.

An End.DX2 function could be customized to expect a specific VLAN format and rewrite the egress VLAN header before forwarding on the outgoing interface.

The End.DX2 is used for L2VPN use-cases.

4.5. End.DX6: Endpoint with decapsulation and IPv6 cross-connect

The "Endpoint with decapsulation and cross-connect to an array of IPv6 adjacencies" function (End.DX6 for short) is a variant of the End and End.X functions.

When N receives a packet destined to S and S is a local End.DX6 SID, N does:

```
1.   IF NH=SRH and SL > 0
2.      drop the packet                                        ;; Ref1
3.   ELSE IF ENH = 41                                         ;; Ref2
4.      pop the (outer) IPv6 header and its extension headers
5.      forward to layer-3 adjacency bound to the SID S        ;; Ref3
6.   ELSE
7.      drop the packet
```

Ref1: The End.DX6 SID must always be the last SID, or it can be the Destination Address of an IPv6 packet with no SRH header.

Ref2: 41 refers to IPv6 encapsulation as defined by IANA allocation for Internet Protocol Numbers

Ref3: Selected based on a hash of the packet’s header (at least SA, DA, Flow Label)

The End.DX6 is used for L3VPN use-cases where a FIB lookup in a specific tenant table at the egress PE is not required. This would be equivalent to the per-CE VPN label in MPLS[RFC4364].

4.6. End.DX4: Endpoint with decapsulation and IPv4 cross-connect

The "Endpoint with decapsulation and cross-connect to an array of IPv4 adjacencies" function (End.DX4 for short) is a variant of the End and End.X functions.

When N receives a packet destined to S and S is a local End.DX4 SID, N does:

```
1.   IF NH=SRH and SL > 0
2.      drop the packet                                    ;; Ref1
3.   ELSE IF ENH  =  4                                     ;; Ref2
4.      pop the (outer) IPv6 header and its extension headers
5.      forward to layer-3 adjacency bound to the SID S    ;; Ref3
6.   ELSE
7.      drop the packet
```

Ref1: The End.DX6 SID must always be the last SID, or it can be the Destination Address of an IPv6 packet with no SRH header.

Ref2: 4 refers to IPv4 encapsulation as defined by IANA allocation for Internet Protocol Numbers

Ref3: Selected based on a hash of the packet’s header (at least SA, DA, Flow Label)

The End.DX4 is used for L3VPN use-cases where a FIB lookup in a specific tenant table at the egress PE is not required. This would be equivalent to the per-CE VPN label in MPLS[RFC4364].

4.7. End.DT6: Endpoint with decapsulation and specific IPv6 table lookup

The "Endpoint with decapsulation and specific IPv6 table lookup" function (End.DT6 for short) is a variant of the End function.

When N receives a packet destined to S and S is a local End.DT6 SID, N does:

```
1.   IF NH=SRH and SL > 0
2.      drop the packet                                    ;; Ref1
3.   ELSE IF ENH = 41                                      ;; Ref2
4.      pop the (outer) IPv6 header and its extension headers
5.      lookup the exposed inner IPv6 DA in IPv6 table T
6.      forward via the matched table entry
7.   ELSE
8.      drop the packet
```

Ref1: the End.DT6 SID must always be the last SID, or it can be the Destination Address of an IPv6 packet with no SRH header.

Ref2: 41 refers to IPv6 encapsulation as defined by IANA allocation for Internet Protocol Numbers

The End.DT6 is used for L3VPN use-cases where a FIB lookup in a specific tenant table at the egress PE is required. This would be equivalent to the per-VRF VPN label in MPLS[RFC4364].

Note that an End.DT6 may be defined for the main IPv6 table in which case and End.DT6 supports the equivalent of an IPv6inIPv6 decaps (without VPN/tenant implication).

4.8. End.DT4: Endpoint with decapsulation and specific IPv4 table lookup

The "Endpoint with decapsulation and specific IPv4 table lookup" function (End.DT4 for short) is a variant of the End function.

When N receives a packet destined to S and S is a local End.DT4 SID, N does:

```
1.   IF NH=SRH and SL > 0
2.      drop the packet                                          ;; Ref1
3.   ELSE IF ENH = 4                                            ;; Ref2
4.      pop the (outer) IPv6 header and its extension headers
5.      lookup the exposed inner IPv4 DA in IPv4 table T
6.      forward via the matched table entry
7.   ELSE
8.      drop the packet
```

Ref1: the End.DT4 SID must always be the last SID, or it can be the Destination Address of an IPv6 packet with no SRH header.

Ref2: 4 refers to IPv4 encapsulation as defined by IANA allocation for Internet Protocol Numbers

The End.DT4 is used for L3VPN use-cases where a FIB lookup in a specific tenant table at the egress PE is required. This would be equivalent to the per-VRF VPN label in MPLS[RFC4364].

4.9. End.B6: Endpoint bound to an SRv6 policy

The "Endpoint bound to an SRv6 Policy" is a variant of the End function.

When N receives a packet destined to S and S is a local End.B6 SID, N does:

```
1.   IF NH=SRH and SL > 0                                        ;; Ref1
2.      do not decrement SL nor update the IPv6 DA with SRH[SL]
3.      insert a new SRH                                         ;; Ref2
4.      set the IPv6 DA to the first segment of the SRv6 Policy
5.      forward according to the first segment of the SRv6 Policy
6.   ELSE
7.      drop the packet
```

Ref1: An End.B6 SID, by definition, is never the last SID.

Ref2: In case that an SRH already exists, the new SRH is inserted in between the IPv6 header and the received SRH

Note: Instead of the term "insert", "push" may also be used.

The End.B6 function is required to express scalable traffic-engineering policies across multiple domains. This is the SRv6 instantiation of a Binding SID [I-D.ietf-spring-segment-routing].

4.10. End.B6.Encaps: Endpoint bound to an SRv6 encapsulation policy

This is a variation of the End.B6 behavior where the SRv6 Policy also includes an IPv6 Source Address A.

When N receives a packet destined to S and S is a local End.B6.Encaps SID, N does:

```
1.   IF NH=SRH and SL > 0
2.      decrement SL and update the IPv6 DA with SRH[SL]
3.      push an outer IPv6 header with its own SRH
4.      set the outer IPv6 SA to A
5.      set the outer IPv6 DA to the first segment of the SRv6 Policy
6.      forward according to the first segment of the SRv6 Policy
7.   ELSE
8.      drop the packet
```

Instead of simply inserting an SRH with the policy (End.B6), this behavior also adds an outer IPv6 header. The source address defined for the outer header does not have to be a local SID of the node.

4.11. End.BM: Endpoint bound to an SR-MPLS policy

The "Endpoint bound to an SR-MPLS Policy" is a variant of the End.B6 function.

When N receives a packet destined to S and S is a local End.BM SID, N does:

```
1.   IF NH=SRH and SL > 0                                    ;; Ref1
2.      decrement SL and update the IPv6 DA with SRH[SL]
3.      push an MPLS label stack <L1, L2, L3> on the received packet
4.      forward according to L1
5.   ELSE
6.      drop the packet
```

Ref1: an End.BM SID, by definition, is never the last SID.

The End.BM function is required to express scalable traffic-engineering policies across multiple domains where some domains support the MPLS instantiation of Segment Routing.

This is an SRv6/SR-MPLS instantiation of a Binding SID [I-D.ietf-spring-segment-routing].

4.12. End.S: Endpoint in search of a target in table T

The "Endpoint in search of a target in Table T" function (End.S for short) is a variant of the End function.

When N receives a packet destined to S and S is a local End.S SID, N does:

```
1.   IF NH=SRH and SL = 0                                    ;; Ref1
2.      drop the packet
3.   ELSE IF match(last SID) in specified table T
4.      forward accordingly
5.   ELSE
6.      apply the End behavior
```

Ref1: By definition, an End.S SID cannot be the last SID, as the last SID is the targeted object.

The End.S function is required in information-centric networking (ICN) use-cases where the last SID in the SRv6 SID list represents a targeted object. If the identification of the object would require more than 128 bits, then obvious customization of the End.S function may either use multiple SIDs or a TLV of the SR header to encode the searched object ID.

4.13. End.AS: Endpoint to SR-unaware APP via static proxy

The "Endpoint to SR-unaware App via Static PROXY" (End.AS for short) is a variant of the End function.

When N receives a packet destined to S and S is a local End.AS SID, it does:

```
1.   IF ENH=4 or ENH=41                                      ;; Ref1
2.      remove the (outer) IPv6 header and its extension headers
3.      forward via the interface associated with the LocalSID  ;; Ref2
4.   ELSE
5.      drop the packet
```

Ref1: 4 and 41 refer to IPv4 encapsulation and IPv6 encapsulation respectively as defined by IANA allocation for Internet Protocol Numbers

Ref2: An SR-unaware app resides in a VM, container or appliance behind this interface. We always assume that the packet that needs to be processed by the app is encapsulated in an outer IPv6 header with its SRH.

The End.AS supports service chaining through SR-unaware application.

When an End.AS SID is locally instantiated at N, it is assumed that the End.AS SID is associated with two interfaces, referred to as target and source interfaces, and an egress SRH. The target interface is the one described above. The source interface is used to recognize the packets coming back from the VM, container or appliance and is associated with an egress SRH. N encapsulates these packets in an outer IPv6 header with the configured egress SRH.

In this scenario, there are no restrictions on the operations that can be performed by the SR-unaware app on the stream of packets. The app can operate at all protocol layers (e.g. it can also terminate transport layer connections); the app can also generate new packets and initiate transport layer connections.

Note that is possible that the target and source interfaces coincide, (i.e. a single interface can be used to send and receive packets to/ from the VM, container or appliance). In this case, the VM, container or appliance can be inserted only in one "unidirectional" chain.

4.14. End.AM: Endpoint to SR-unaware APP via masquerading

The "Endpoint to SR-unaware App via Masquerading" (End.AM for short) is a variant of the End function.

When N receives a packet destined to S and S is a local End.AM SID, it does:

```
1.   IF NR=SRH & SL > 0                                          ;; Ref1
2.      decrement SL
3.      write the last SID in the IPv6 DA
4.      forward via the interface associated with the LocalSID   ;; Ref2
5.   ELSE
6.      drop the packet
```

Ref1: An End.AM must not be the last SID.

Ref2: An SR-unaware VNF resides behind this interface

The End.AM supports service chaining through SR-unaware application.

We "masquerade" the packet for two reasons:

1.- We want the app to receive a packet with the source and destination addresses respectively set as the true source and the final destination.

2.- We do not want the app to support/read the SRH. We leverage [RFC2460] which specifies that a transit node does not need to process an IPv6 routing extension header.

When an End.AM SID is locally instantiated at N, it is assumed that two interfaces are associated with the SID, referred to as target and source. The target interface is the one described above. The source interface identifies the packets coming back from the app. N is set to always inspect the SRH of the packets coming from the source interface and set their DA = SRH[SL] (to "de-masquerade" the SRH header).

In this scenario, we assume that the app residing in the VM, container or appliance can only inspect the packets, drop the packets, perform limited changes to the packet (in particular, the app must not change the IP Destination Address of the packet). The app cannot terminate a transport connection nor generate arbitrary packets. For example Firewalls, Intrusion Detection Systems, Deep Packet Inspectors are among the app classes that can be supported in this scenario.

4.15. SR-aware application

Generally, any SR-aware application can be bound to an SRv6 SID. This application could represent anything from a small piece of code focused on topological/tenant function to a much larger process focusing on higher-level applications (e.g. video compression, transcoding etc.).

The ways in which an SR-aware application can binds itself on a local SID depends on the operating system. Let us consider an SR-aware application running on a Linux operating system. A possible approach is to associate an SRv6 SID to a target (virtual) interface, so that packets with IP DA corresponding to the SID will be sent to the target interface. In this approach, the SR-aware application can simply listen to all packets received on the interface.

A different approach for the SR-aware app is to listen to packets received with a specific SRv6 SID as IPv6 DA on a given transport port (i.e. corresponding to a TCP or UDP socket). In this case, the app can read the SRH information with a getsockopt Linux system call and can set the SRH information to be added to the outgoing packets with a setsocksopt system call.

4.16. Flavours

We present the PSP and USP variants of the functions End, End.X and End.T. For each of these functions these variants can be enabled or disabled either individually or together.

4.16.1. PSP: penultimate segment POP of the SRH

After the instruction 'update the IPv6 DA with SRH[SL]' is executed, the following instructions must be added:

```
1.   IF updated SL = 0 & PSP is TRUE
2.      pop the top SRH                                          ;; Ref1
```

Ref1: The received SRH had SL=1. When the last SID is written in the DA, the End, End.X and End.T functions with the PSP flavour pop the first (top-most) SRH. Subsequent stacked SRH's may be present but are not processed as part of the function.

4.16.2. USP: Ultimate Segment Pop of the SRH

We insert at the beginning of the pseudo-code the following instructions:

```
1.   IF SL = 0 & NH=SRH & USP=TRUE                               ;; Ref1
2.      pop the top SRH
3.      restart the function processing on the modified packet   ;; Ref2
```

Ref1: The next header is an SRH header

Ref2: Typically SL of the exposed SRH is > 0 and hence the restarting of the complete function would lead to decrement SL, update the IPv6 DA with SRH[SL], FIB lookup on updated DA and forward accordingly to the matched entry.

5. Transit behaviors

We define hereafter the set of basic transit behaviors.

T	Transit behavior
T.Insert	Transit behavior with insertion of an SRv6 Policy
T.Encaps	Transit behavior with encapsulation in an SRv6 policy
T.Encaps.L2	T.Encaps behavior of the received L2 frame

This list can be expanded in case any new functionality requires it.

5.1. T: Transit behavior

As per [RFC2460], if a node N receives a packet (A, S2)(S3, S2, S1; SL=2) and S2 is neither a local address nor a local SID of N then N forwards the packet without inspecting the SRH.

This means that N treats the following two packets with the same performance:

- (A, S2)

- (A, S2)(S3, S2, S1; SL=2)

A transit node does not need to count by default the amount of transit traffic with an SRH extension header. This accounting might be enabled as an optional behavior leveraging SEC4 behavior described later in this document.Section 7.4

A transit node MUST include the outer flow label in its ECMP hash[RFC6437].

5.2. T.Insert: Transit with insertion of an SRv6 Policy

Node N receives two packets P1=(A, B2) and P2=(A,B2)(B3, B2, B1; SL=1). B2 is neither a local address nor SID of N.

N steers the transit packets P1 and P2 into an SRv6 Policy with one SID list <S1, S2, S3>.

The "T.Insert" transit insertion behavior is defined as follows:

1. insert the SRH (B2, S3, S2, S1; SL=3) ;; Ref1, Ref1bis
2. set the IPv6 DA = S1
3. forward along the shortest path to S1

Ref1: The received IPv6 DA is placed as last SID of the inserted SRH.

Ref1bis: The SRH is inserted before any other IPv6 Routing Extension Header.

After the T.Insert behavior, P1 and P2 respectively look like:

- (A, S1) (B2, S3, S2, S1; SL=3)

- (A, S1) (B2, S3, S2, S1; SL=3) (B3, B2, B1; SL=1)

5.3. T.Encaps: Transit with encapsulation in an SRv6 Policy

Node N receives two packets P1=(A, B2) and P2=(A,B2)(B3, B2, B1; SL=1). B2 is neither a local address nor SID of N.

N steers the transit packets P1 and P2 into an SR Encapsulation Policy with a Source Address A and a Segment list <S1, S2, S3>.

The T.Encaps transit encapsulation behavior is defined as follows:

```
1.   push an IPv6 header with its own SRH (S3, S2, S1; SL=2)
2.   set outer IPv6 SA = N and outer IPv6 DA = S1
3.   set outer payload length, traffic class and flow label     ;; Ref 1
4.   update the next_header value                               ;; Ref 1
5.   decrement inner Hop Limit or TTL                          ;; Ref 1
6.   forward along the shortest path to S1
```

After the T.Encaps behavior, P1 and P2 respectively look like:

- (T, S1) (S3, S2, S1; SL=2) (A, B2)

- (T, S1) (S3, S2, S1; SL=2) (A, B2) (B3, B2, B1; SL=1)

The T.Encaps behavior is valid for any kind of Layer-3 traffic. This behavior is commonly used for L3VPN with IPv4 and IPv6 deployements.

The SRH MAY be omitted when the SRv6 Policy only contains one segment and there is no need to use any flag, tag or TLV.

Ref 1: As described in [RFC2473] (Generic Packet Tunneling in IPv6 Specification)

5.4. T.Encaps.L2: Transit with encapsulation of L2 frames

While T.Encaps encapsulates the received IP packet, T.Encaps.L2 encapsulates the received L2 frame (i.e. the received ethernet header and its optional VLAN header is in the payload of the outer packet).

If the outer header is pushed without SRH then the DA must be a SID of type End.DX2 and the next-header must be 59 (IPv6 NoNextHeader). The received Ethernet frame follows the IPv6 header and its extension headers.

Else, if the outer header is pushed with an SRH, then the last SID of the SRH must be of type End.DX2 and the next-header of the SRH must be 59 (IPv6 NoNextHeader). The received Ethernet frame follows the IPv6 header and its extension headers.

6. Operation

6.1. Counters

Any SRv6 capable node MUST implement the following set of combined counters (packets and bytes):

- Per entry of the "My Local SID Table":
  - Traffic that matched that SID and was processed correctly
  - Traffic that matched that SID and was NOT processed correctly
- Per SRv6 Policy:
  - Traffic steered into it and processed correctly
  - Traffic steered into it and NOT processed correctly

Furthermore, an SRv6 capable node maintains an aggregate counter tracking the IPv6 traffic that was received with a destination address matching a local interface address that is not a local SID and the next-header is SRH. We remind that this traffic is dropped as an interface address is not a local SID by default. A SID must be explicitly instantiated.

6.2. Flow-based hash computation

When a flow-based selection within a set needs to be performed, the source address, the destination address and the flow-label MUST be included in the flow-based hash.

This occurs when the destination address is updated and a FIB lookup is performed and multiple ECMP paths exist to the updated destination address.

This occurs when End.X is bound to an array of adjacencies.

This occurs when the packet is steered in an SR policy whose selected path has multiple SID lists [I-D.filsfils-spring-segment-routing-policy].

7. Basic security for intra-domain deployment

We use the following terminology:

An internal node is a node part of the domain of trust.

A border router is an internal node at the edge of the domain of trust.

An external interface is an interface of a border router towards another domain.

An internal interface is an interface entirely within the domain of trust.

The internal address space is the IP address block dedicated to internal interfaces.

An internal SID is a SID instantiated on an internal node.

The internal SID space is the IP address block dedicated to internal SIDs.

External traffic is traffic received from an external interface to the domain of trust.

Internal traffic is traffic the originates and ends within the domain of trust.

The purpose of this section is to document how a domain of trust can operate SRv6-based services for internal traffic while preventing any external traffic from accessing the internal SRv6-based services.

It is expected that future documents will detail enhanced security mechanisms for SRv6 (e.g. how to allow external traffic to leverage internal SRv6 services).

7.1. SEC 1

An SRv6 router MUST support an ACL on the external interface that drops any traffic with SA or DA in the internal SID space.

A provider would generally do this for its internal address space to prevent access to internal addresses and in order to prevent spoofing. The technique is extended to the local SID space.

The typical counters of an ACL are expected.

7.2. SEC 2

An SRv6 router MUST support an ACL with the following behavior:

```
1.   IF (DA == LocalSID) && (SA != internal address or SID space)
2.      drop
```

This prevents access to local SIDs from outside the operator's infrastructure. Note that this ACL may not be enabled in all cases. For example, specific SIDs can be used to provide resources to devices that are outside of the operator's infrastructure.

When an SID is in the form of LOC:FUNCT:ARGS the DA match should be implemented as a prefix match covering the argument space of the specific SID i.s.o. a host route.

The typical counters of an ACL are expected.

7.3. SEC 3

As per the End definition, an SRv6 router MUST only implement the End behavior on a local IPv6 address if that address has been explicitly enabled as a segment.

This address may or may not be associated with an interface. This address may or may not be routed. The only thing that matters is that the local SID must be explicitly instantiated and explicitly bound to a function (the default function is the End function).

7.4. SEC 4

An SRv6 router should support Unicast-RPF on source address on external interface.

This is a generic provider technique applied to the internal address space. It is extended to the internal SID space.

The typical counters to validate such filtering are expected.

8. Control Plane

In an SDN environment, one expects the controller to explicitly provision the SIDs and/or discover them as part of a service discovery function. Applications residing on top of the controller could then discover the required SIDs and combine them to form a distributed network program.

The concept of "SRv6 network programming" refers to the capability for an application to encode any complex program as a set of individual functions distributed through the network. Some functions relate to underlay SLA others to overlay/tenant, others to complex applications residing in VM and containers.

The specification of the SRv6 control-plane is outside the scope of this document.

We limit ourselves to a few important observations.

8.1. IGP

The End and End.X SIDs express topological functions and hence are expected to be signaled in the IGP together with the flavours PSP and USP.

The presence of SIDs in the IGP do not imply any routing semantics to the addresses represented by these SIDs. The routing reachability to an IPv6 address is solely governed by the classic, non-SID-related, IGP information. Routing is not governed neither influenced in any way by a SID advertisement in the IGP.

These two SIDs provide important topological functions for the IGP to build FRR/TI-LFA solution and for TE processes relying on IGP LSDB to build SR policies.

8.2. BGP-LS

BGP-LS is expected to be the key service discovery protocol. Every node is expected to advertise via BGP-LS its SRv6 capabilities (e.g. how many SIDs in can insert as part of an T.Insert behavior) and any locally instantiated SID[I-D.ietf-idr-bgp-ls-segment-routing-ext][I-D.ietf-idr-te-lsp-distribution].

8.3. BGP IP/VPN

The End.DX46, End.DT46 and End.DX2 SIDs are expected to be signaled in BGP.

8.4. Summary

The following table summarizes which SID would be signaled in which signaling protocol.

	IGP	BGP-LS	BGP IP/VPN
End (PSP, USP)	X	X	
End.X (PSP, USP)	X	X	
End.T (PSP, USP)	X	X	
End.DX2		X	X
End.DX6		X	X
End.DX4		X	X
End.DT6		X	X
End.DT4		X	X
End.B6		X	
End.B6.Encaps		X	
End.B6.BM		X	
End.S		X	
End.AS		X	
End.AM		X	

Table 1: SRv6 LocalSID signaling

The following table summarizes which transit capability would be signaled in which signaling protocol.

	IGP	BGP-LS	BGP IP/VPN
T		X	
T.Insert		X	
T.Encaps		X	
T.Encaps.L2		X	

Table 2: SRv6 transit behaviors signaling

The previous table describes generic capabilities. It does not describe specific instantiated SID.

For example, a BGP-LS advertisement of the T capability of node N would indicate that node N supports the basic transit behavior. The T.Insert behavior would describe the capability of node N to instantiation a T.Insert behavior, specifically it would describe how many SIDs could be inserted by N without significant performance degradation. Same for T.Encaps (the number potentially lower as the overhead of the additional outer IP header is accounted).

The reader should also remember that any specific instantiated SR policy (via T.Insert or T.Encaps) is always assigned a Binding SID. He should remember that BSIDs are advertised in BGP-LS as shown in Table 1. Hence, it is normal that Table 2 only focuses on the generic capabilities related to T.Insert and T.Encaps as Table 1 advertises the specific instantiated BSID properties.

9. Illustration

We introduce a simplified SID allocation technique to ease the reading of the text. We document the reference diagram. We then illustrate the network programming concept through different use-cases. These use-cases have been thought to allow straightforward combination between each other.

9.1. Simplified SID allocation

To simplify the illustration, we assume:

A::/4 is dedicated to the internal SRv6 SID space

B::/4 is dedicated to the internal address space

We assume a location expressed in 48 bits and a function expressed in 80 bits

Node k has a classic IPv6 loopback address Bk::/128 which is advertised in the IGP

Node k has Ak::/48 for its local SID space. Its SIDs will be explicitly allocated from that block

Node k advertises Ak::/48 in its IGP

Function 0:0:0:0:0 (function 0, for short) represents the End function

Function 0:0:0:0:C2 (function C2, for short) represents the End.X function towards neighbor 2

Function 0:0:0:0:E100 (function E100, for short) represents the End.T function in tenant table 100

Each node K has:

An explicit SID instantiation Ak::0/128 bound to an End function with additional support for PSP and USP

An explicit SID instantiation Ak::Cj/128 bound to an End.X function to neighbor J with additional support for PSP and USP

9.2. Reference diagram

Let us assume the following topology where all the links have IGP metric 10 except the link 23 which is 100.

Nodes A, 1 to 8 and B are considered within the network domain while nodes CE-A and CE-B are outside the domain.

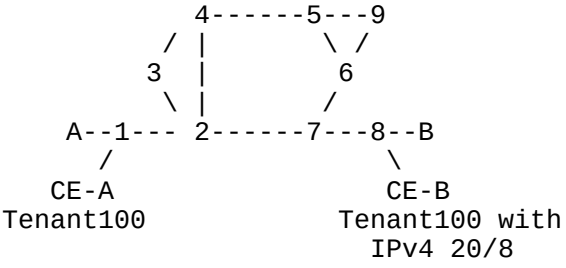

Figure 1: Reference topology

9.3. Basic security

Any edge node such as 1 would be configured with an ACL on any of its external interface (e.g. from CE-A) which drops any traffic with SA or DA in A::/4. See SEC 1 (Section 7.1).

Any core node such as 6 could be configured with an ACL with the SEC2 (Section 7.2) behavior "IF (DA == LocalSID) && (SA is not in A::/4 or B::/4) THEN drop".

SEC 3 (Section 7.3) protection is a default property of SRv6. A SID must be explicitly instantiated. In our illustration, the only available SIDs are those explicitly instantiated.

Any edge node such as 1 would be configured with Unicast-RPF on source address on external interface (e.g. from CE-A). See SEC 4 (Section 7.4).

9.4. SR-IPVPN

Let us illustrate the SR-IPVPN use-case applied to IPv4.

Nodes 1 and 8 are configured with a tenant 100, each respectively connected to CE-A and CE-B.

Node 8 is configured with a local SID A8::E100 of function End.DT4(100) bound to tenant IPv4 table 100.

Via BGP signaling or an SDN-based controller, Node 1's tenant-100 IPv4 table is programmed with an IPv4 SR-VPN route 20/8 via SRv6 policy <A8::E100>.

When 1 receives a packet P from CE-A destined to 20.20.20.20, P looks up its tenant-100 IPv4 table and finds an SR-VPN entry 20/8 via SRv6 policy <A8::E100>. As a consequence, 1 pushes an outer IPv6 header with SA=A1::0, DA=A8::E100 and NH=4. 1 then forwards the resulting packet on the shortest path to A8::/40.

When 8 receives the packet, 8 matches the DA in its My LocalSID table, finds the bound function End.DT4(100) and confirms NH=4. As a result, 8 decaps the outer header, looks up the inner IPv4 DA in tenant-100 IPv4 table, and forward the (inner) IPv4 packet towards CE-B.

The reader can easily infer all the other SR-IPVPN IP instantiations:

Route at ingress PE(1)	SR-VPN Egress SID of egress PE(8)
IPv4 tenant route with egress tenant table lookup	End.DT4 function bound to IPv4-tenant-100 table
IPv4 tenant route without egress tenant table lookup	End.DX4 function bound to CE-B (IPv4)
IPv6 tenant route with egress tenant table lookup	End.DT6 function bound to IPv6-tenant-100 table
IPv6 tenant route without egress tenant table lookup	End.DX6 function bound to CE-B (IPv6)

9.5. SR-Ethernet-VPWS

Let us illustrate the SR-Ethernet-VPWS use-case.

Node 1 is configured with an ethernet VPWS service:

- Local attachment circuit: Ethernet interface from CE-A

- Local End.DX2 bound to the local attachment circuit: A1::C2A

- Remote End.DX2 SID: A8::C2B

Node 8 is configured with an ethernet VPWS service:

- Local attachment circuit: Ethernet interface from CE-B

- Local End.DX2 bound to the local attachment circuit: A8::C2B

- Remote End.DX2 SID: A1::C2A

These configurations can either be programmed by an SDN controller or partially derived from a BGP-based signaling and discovery service .

When 1 receives a packet P from CE-A, 1 pushes an outer IPv6 header with SA=A1::0, DA=A8::C2B and NH=59. Note that no additional header is pushed. 1 then forwards the resulting packet on the shortest path to A8::/40.

When 8 receives the packet, 8 matches the DA in its My LocalSID table and finds the bound function End.DX2. After confirming that the next-header=59, 8 decaps the outer IPv6 header and forwards the inner Ethernet frame towards CE-B.

The reader can easily infer the Ethernet multipoint use-case:

Route at ingress PE(1)	SR-VPN Egress SID of egress PE(8)
Ethernet VPWS	End.DX2 function bound to CE-B (Ethernet)

9.6. SR TE for Underlay SLA

9.6.1. SR policy from the Ingress PE

Let’s assume that node 1’s tenant-100 IPv4 route "20/8 via A8::E100" is programmed with a color/community that requires low-latency underlay optimization [I-D.filsfils-spring-segment-routing-policy].

In such case, node 1 either computes the low-latency path to the egress node itself or delegates the computation to a PCE.

In either case, the location of the egress PE can easily be found by looking for who originates the SID block comprising the SID A8::E100. This can be found in the IGP’s LSDB for a single domain case, and in the BGP-LS LSDB for a multi-domain case.

Let us assume that the TE metric encodes the per-link propagation latency. Let us assume that all the links have a TE metric of 10, except link 27 which has TE metric 100.

The low-latency path from 1 to 8 is thus 1245678.

This path is encoded in a SID list as: first a hop through A4::C5 and then a hop to 8.

As a consequence the SR-VPN entry 20/8 installed in the Node1's Tenant-100 IPv4 table is: T.Encaps with SRv6 Policy <A4::C5, A8::E100>.

When 1 receives a packet P from CE-A destined to 20.20.20.20, P looks up its tenant-100 IPv4 table and finds an SR-VPN entry 20/8. As a consequence, 1 pushes an outer header with SA=A1::0, A4::C5, NH=SRH followed by SRH (A8::E100, A4::C5; SL=1; NH=4) . 1 then forwards the resulting packet on the interface to 2.

2 forwards to 4 along the path to A4::/40.

When 4 receives the packet, 4 matches the DA in its My LocalSID table and finds the bound function End.X to neighbor 5. 4 notes the PSP capability of the SID A4::C5. 4 sets the DA to the next SID A8::E100. As 4 is the penultimate segment hop, it performs PSP and pops the SRH. 4 forwards the resulting packet to 5.

5, 6 and 7 forwards along the path to A8::/40.

When 8 receives the packet, 8 matches the DA in its My LocalSID Table and finds the bound function End.DT(100). As a result, 8 decaps the outer header, looks up the inner IPv4 DA in tenant-100 IPv4 table, and forward the (inner) IPv4 packet towards CE-B.

9.6.2. SR policy at a midpoint

Let us analyze a policy applied at a midpoint on a packet without SRH.

Packet P1 is (A1::, A8::E100).

Let us consider P1 when it is received by node 2 and let us assume that that node 2 is configured to steer A8::/40 in a transit behavior T.Insert associated with SR policy <A4::C5>.

In such a case, node 2 would send the following modified packet P1 on the link to 4:

(A1::, A4::C5)(A8::E100, A4::C5; SL=1).

The rest of the processing is similar to the previous section.

Let us analyze a policy applied at a midpoint on a packet with an SRH.

Packet P2 is (A1::, A7::)(A8::E100, A7::; SL=1).

Let us consider P2 when it is received by node 2 and let us assume that node 2 is configured to steer A7::/40 in a transit behavior T.Insert associated with SR policy <A4::C5, A9::>.

In such a case, node 2 would send the following modified packet P2 on the link to 4:

(A1::, A4::C5)(A7::, A9::, A4::C5; SL=2)(A8::E100, A7::; SL=1)

Node 4 would send the following packet to 5: (A1::, A9::)(A7::, A9::, A4::C5; SL=1)(A8::E100, A7::; SL=1)

Node 5 would send the following packet to 9: (A1::, A9::)(A7::, A9::, A4::C5; SL=1)(A8::E100, A7::; SL=1)

Node 9 would send the following packet to 6: (A1::, A7::)(A8::E100, A7::; SL=1)

Node 6 would send the following packet to 7: (A1::, A7::)(A8::E100, A7::; SL=1)

Node 7 would send the following packet to 8: (A1::, A8::E100)

9.7. End-to-End policy with intermediate BSID

Let us now describe a case where the ingress VPN edge node steers the packet destined to 20.20.20.20 towards the egress edge node connected to the tenant100 site with 20/8, but via an intermediate SR Policy represented by a single routable Binding SID. Let us illustrate this case with an intermediate policy which both encodes underlay optimization for low-latency and the service chaining via two SR-aware container-based apps.

Let us assume that the End.B6 SID A2::B1 is configured at node 2 and is associated with midpoint T.Insert policy <A4::C5, A9::A1, A6::A2>.

A4::C5 realizes the low-latency path from the ingress PE to the egress PE. This is the underlay optimization part of the intermediate policy.

A9::A1 and A6::A2 represent two SR-aware NFV applications residing in containers respectively connected to node 9 and 6.

Let us assume the following ingress VPN policy for 20/8 in tenant 100 IPv4 table of node 1: T.Encaps with SRv6 Policy <A2::B1, A8::E100>.

This ingress policy will steer the 20/8 tenant-100 traffic towards the correct egress PE and via the required intermediate policy that realizes the SLA and NFV requirements of this tenant customer.

Node 1 sends the following packet to 2: (A1::, A2::B1) (A8::E100, A2::B1; SL=1)

Node 2 sends the following packet to 4: (A1::, A4::C5) (A6::A2, A9::A1, A4::C5; SL=2)(A8::E100, A2::B1; SL=1)

Node 4 sends the following packet to 5: (A1::, A9::A1) (A6::A2, A9::A1, A4::C5; SL=1)(A8::E100, A2::B1; SL=1)

Node 5 sends the following packet to 9: (A1::, A9::A1) (A6::A2, A9::A1, A4::C5; SL=1)(A8::E100, A2::B1; SL=1)

Node 9 sends the following packet to 6: (A1::, A6::A2) (A8::E100, A2::B1; SL=1)

Node 6 sends the following packet to 7: (A1::, A8::E100)

Node 7 sends the following packet to 8: (A1::, A8::E100) which decaps and forwards to CE-B.

The benefits of using an intermediate Binding SID are well-known and key to the Segment Routing architecture: the ingress edge node needs to push fewer SIDs, the ingress edge node does not need to change its SR policy upon change of the core topology or re-homing of the container-based apps on different servers. Conversely, the core and service organizations do not need to share details on how they realize underlay SLA’s or where they home their NFV apps.

9.8. TI-LFA

Let us assume two packets P1 and P2 received by node 2 exactly when the failure of link 27 is detected.

P1: (A1::, A7::)

P2: (A1::, A7::)(A8::E100, A7::; SL=1)

Node 2's pre-computed TI-LFA backup path for the destination A7:: is <A4::C5>. It is installed as a T.Insert transit behavior.

Node 2 protects the two packets P1 and P2 according to the pre-computed TI-LFA backup path and send the following modified packets on the link to 4:

P1: (A1::, A4::C5)(A7::, A4::C5; SL=1)

P2: (A1::, A4::C5)(A7::, A4::C5; SL=1) (A8::E100, A7::; SL=1)

Node 4 then sends the following modified packets to 5:

P1: (A1::, A7::)

P2: (A1::, A7::)(A8::E100, A7::; SL=1)

Then these packets follow the rest of their post-convergence path towards node 7 and then go to node 8 for the VPN decaps.

9.9. SR TE for Service chaining

We have illustrated the service chaining through SR-aware apps in a previous section.

We illustrate the use of End.AS functions to service chain an IP flow bound to the internet through two SR-unaware applications hosted in containers.

Let us assume that servers 20 and 70 are respectively connected to nodes 2 and 7. They are respectively configured with SID spaces A020::/40 and A070::/40. Their connected routers advertise the related prefixes in the IGP. Two SR-unaware container-based applications App2 and App7 are respectively hosted on server 20 and 70. Server 20 (70) is configured explicitly with an End.AS SID A020::2 for App2 (A070::7 for App7).

Let us assume a broadband customer with a home gateway CE-A connected to edge router 1. Router 1 is configured with an SR policy which encapsulates all the traffic received from CE-A into a T.Encaps policy <A020::2, A070::7, A8::E0> where A8::E0 is an End.DT4 SID instantiated at node 8.

P1 is a packet sent by the broadband customer to 1: (X, Y) where X and Y are two IPv4 addresses.

1 sends the following packet to 2: (A1::0, A020::2)(A8::E0, A070::7, A020::2; SL=2; NH=4)(X, Y).

2 forwards the packet to server 20.

20 recievs the packet (A1::0, A070::7)(A8::E0, A070::7, A020::2; SL=2; NH=1)(X, Y) and forwards the inner IPv4 packet (X,Y) to App2. App2 works on the packet and forwards it back to 20. 20 sends the (whole) IPv6 packet back to 2.

2 and 7 forward to server 70.

70 recieves the packet (A1::0, A8::E0)(X, Y) and forwards the inner IPv4 packet (X,Y) to App7. App7 works on the packet and forwards it back to 70. 70 sends the (whole) IPv6 packet back to 7.

7 forwards to 8.

8 performs the End.DT4 function and sends the IP packet (X, Y) towards its internet destination

10. Benefits

10.1. Seamless deployment

The VPN use-case can be realized with SRv6 capability deployed solely at the ingress and egress PE’s.

All the nodes in between these PE’s act as transit routers as per [RFC2460]. No software/hardware upgrade is required on all these nodes. They just need to support IPv6 per [RFC2460].

The SRTE/underlay-SLA use-case can be realized with SRv6 capability deployed at few strategic nodes.

It is well-known from the experience deploying SR-MPLS that underlay SLA optimization requires few SIDs placed at strategic locations. This was illustrated in our example with the low-latency optimization which required the operator to enable one single core node with SRv6 (node 4) where one single and End.X SID towards node 5 was instantiated. This single SID is sufficient to force the end-to-end traffic via the low-latency path.

The TI-LFA benefits are collected incrementally as SRv6 capabilities are deployed.

It is well-know that TI-LFA is an incremental node-by-node deployment. When a node N is enabled for TI-LFA, it computes TI-

LFA backup paths for each primary path to each IGP destination. In more than 50% of the case, the post-convergence path is loop-free and does not depend on the presence of any remote SRv6 SID. In the vast majority of cases, a single segment is enough to encode the post-convergence path in a loop-free manner. If the required segment is available (that node has been upgraded) then the related back-up path is installed in FIB, else the pre-existing situation (no backup) continues. Hence, as the SRv6 deployment progresses, the coverage incrementally increases. Eventually, when the core network is SRv6 capable, the TI-LFA coverage is complete.

The service chaining use-case can be realized with SRv6 capability deployed at few strategic nodes.

The service-chaining deployment is again incremental and does not require any pre-deployment of SRv6 in the network. When an NFV app A1 needs to be enabled for inclusion in an SRv6 service chain, all what is required is to install that app in a container or VM on an SRv6-capable server (Linux 4.10 or FD.io 17.04 release). The app can either be SR-aware or not, leveraging the proxy functions described in this document.

By leveraging the various END functions it can also be used to support any current PNF/VNF implementations and their forwarding methods (e.g. Layer 2).

The ability to leverage SR TE policies and BSIDs also permits building scalable, hierarchical service-chains.

10.2. Integration

The SRv6 network programming concept allows integrating all the application and service requirements: multi-domain underlay SLA optimization with scale, overlay VPN/Tenant, sub-50msec automated FRR, security and service chaining.

10.3. Security

The combination of well-known techniques (SEC1, SEC2, SEC4) and carefully chosen architectural rules (SEC3) ensure a secure deployment of SRv6 inside a multi-domain network managed by a single organization.

Inter-domain security will be described in a companion document.

11. IANA Considerations

This document has no actions for IANA.

12. Acknowledgements

TBD.

13. Contributors

Stefano Previdi, Dave Barach, Mark Townsley, Peter Psenak, Paul Wells, Robert Hanzl, Dan Ye, Patrice Brissette, Gaurav Dawra, Faisal Iqbal, Zafar Ali, Jaganbabu Rajamanickam, David Toscano, Asif Islam, Jianda Liu, Yunpeng Zhang, Jiaoming Li, Narendra A.K, Mike Mc Gourty, Bhupendra Yadav, Sherif Toulan, Satish Damodaran, John Bettink, Kishore Nandyala Veera Venk, Jisu Bhattacharya and Saleem Hafeez substantially contributed to the content of this document.

14. References

14.1. 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>.

14.2. Informative References

[I-D.filsfils-spring-segment-routing-policy] Filsfils, C., Sivabalan, S., Yoyer, D., Nanduri, M., Lin, S., bogdanov@google.com, b., Horneffer, M., Clad, F., Steinberg, D., Decraene, B., and S. Litkowski, "Segment Routing Policy for Traffic Engineering", draft-filsfils-spring-segment-routing-policy-00 (work in progress), February 2017.

[I-D.ietf-6man-segment-routing-header] Previdi, S., Filsfils, C., Field, B., Leung, I., Linkova, J., Aries, E., Kosugi, T., Vyncke, E., and D. Lebrun, "IPv6 Segment Routing Header (SRH)", draft-ietf-6man-segment-routing-header-05 (work in progress), February 2017.

[I-D.ietf-idr-bgp-ls-segment-routing-ext]
Previdi, S., Psenak, P., Filsfils, C., Gredler, H., Chen, M., and j. jefftant@gmail.com, "BGP Link-State extensions for Segment Routing", draft-ietf-idr-bgp-ls-segment-routing-ext-01 (work in progress), February 2017.

[I-D.ietf-idr-te-lsp-distribution]
Previdi, S., Dong, J., Chen, M., Gredler, H., and j. jefftant@gmail.com, "Distribution of Traffic Engineering (TE) Policies and State using BGP-LS", draft-ietf-idr-te-lsp-distribution-06 (work in progress), January 2017.

[I-D.ietf-isis-l2bundles]
Ginsberg, L., Bashandy, A., Filsfils, C., Previdi, S., Nanduri, M., and E. Aries, "Advertising L2 Bundle Member Link Attributes in IS-IS", draft-ietf-isis-l2bundles-03 (work in progress), February 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-11 (work in progress), February 2017.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473, December 1998, <http://www.rfc-editor.org/info/rfc2473>.

[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>.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <http://www.rfc-editor.org/info/rfc6437>.

Authors' Addresses

Clarence Filsfils
Cisco Systems, Inc.
Belgium

Email: cf@cisco.com

John Leddy
Comcast
United States of America

Email: john_leddy@cable.comcast.com

Daniel Voyer
Bell Canada
Canada

Email: daniel.voyer@bell.ca

Daniel Bernier
Bell Canada
Canada

Email: daniel.bernier@bell.ca

Dirk Steinberg
Steinberg Consulting
Germany

Email: dws@dirksteinberg.de

Robert Raszuk
Bloomberg LP
United States of America

Email: robert@raszuk.net

Satoru Matsushima
SoftBank Telecom
Japan

Email: satoru.matsushima@g.softbank.co.jp

David Lebrun
Universite catholique de Louvain
Belgium

Email: david.lebrun@uclouvain.be

Bruno Decraene
Orange
France

Email: bruno.decraene@orange.com

Bart Peirens
Proximus
Netherlands

Email: bart.peirens@proximus.com

Stefano Salsano
Universita di Roma "Tor Vergata"
Italy

Email: stefano.salsano@uniroma2.it

Gaurav Naik
Drexel University
United States of America

Email: gn@drexel.edu

Hani Elmalky
Ericsson
United States of America

Email: hani.elmalky@ericsson.com

Prem Jonnalagadda
Barefoot Networks
United States of America

Email: prem@barefootnetworks.com

Milad Sharif
Barefoot Networks
United States of America

Email: msharif@barefootnetworks.com

Arthi Ayyangar
Arista
United States of America

Email: arthi@arista.com

Satish Mynam
Dell Force10 Networks
United States of America

Email: satish_mynam@dell.com

Ahmed Bashandy
Cisco Systems, Inc.
United States of America

Email: bashandy@cisco.com

Kamran Raza
Cisco Systems, Inc.
Canada

Email: skraza@cisco.com

Darren Dukes
Cisco Systems, Inc.
Canada

Email: ddukes@cisco.com

Francois Clad
Cisco Systems, Inc.
France

Email: fclad@cisco.com

Pablo Camarillo Garvia (editor)
Cisco Systems, Inc.
Spain

Email: pcamaril@cisco.com

SPRING
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

S. Hegde
Juniper Networks, Inc.
A. Gulko
Thomson Reuters

March 13, 2017

Separating Routing Planes using Segment Routing
draft-gulkohegde-routing-planes-using-sr-00

Abstract

Many network deployments arrange the network topologies in two or more planes. The traffic generally uses one of the planes and fails over to the other plane when there are link or node failure. Certain applications require the traffic to be strictly restricted to a particular plane and should not failover to the other plane. This document proposes a solution for the strict planar routing using Segment Routing.

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 14, 2017.

Table of Contents

1. Introduction

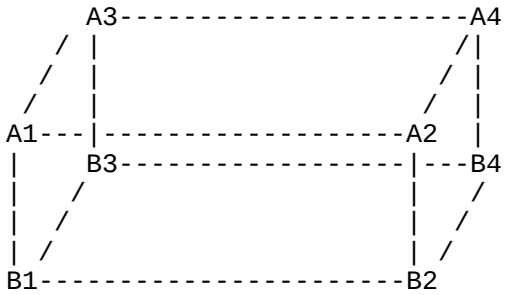

Figure 1: Network Planes

The above Figure 1 represents a network topology in two planes.Nodes A1, A2, A3, A4 are in plane A and B1, B2, B3, B4 are in plane B. A1->B1, A2->B2, A3->B3, A4->B4 represent the "shunt links" which connect the two planes. Certain applications require that the

traffic follows plane A and remains in plane A in case of failures and does not cross over to plane B. Strict routing plane requirements can be met using multiple techniques. This draft focuses on solutions using Segment Routing technology.

2. Motivation

The motivation of this document is to provide solutions to strict routing plane requirements using Segment Routing. The following objectives help to accomplish this in a range of deployment scenarios.

1. Maintain strict routing within routing planes.

2. Allow traffic to failover within routing plane and do not allow traffic to failover to other planes

3. Achieve ease of configuration and operational management

3. Solutions

There are multiple ways to address the strict routing plane requirements. Section 3.1 describes a mechanism by using diferrent SIDs for each plane.Another option is to use Multi-topology SIDs as defined in Section 3.2.

3.1. Routing-plane SID

A new SID called Routing-Plane SID is defined and associated with new algorithm values. This document proposes 4 new algorithm values which represent SPF in routing-planes. When the network topology is organized into two different planes, each node in plane A associates a new Routing-Plane SID to one of it’s loopback addresses and advertises the SID in the segment routing extension defined in [SR-OSPF] section 2.1 and [SR-IS-IS] section 5. The prefix-SID sub-TLV carries the new algorithm values corresponding to the routing-plane. The traffic which needs to be restricted to a certain routing-plane,should use the Routing-Plane SID corresponding to that plane to forward the traffic. The paths through the Routing planes MAY use single Routing Plane SID or a stack of multiple Routing Plane SIDs. Adjacency-SIDs can also be used build paths across routing planes. Adjacency-SIDs are not scoped per-algorithm and it is possible that the protection path for the adjacency SIDs uses a path going over a different routing-plane.It is recommended to use non-protected adjacency-SIDs to avoid backup traffic flowing through different plane.

3.1.1. Elements of procedure

All the nodes that belong to a certain routing-plane MUST advertise the algorithm corresponding to that routing-plane in the algorithm sub-TLV as defined in [SR-OSPF] and [SR-IS-IS]. The nodes SHOULD also advertise Routing-Plane SID corresponding to that algorithm in the prefix-SID Sub-TLV.

A node that receives the algorithm sub-TLV with new algorithm value specified in Section 6 MUST compute SPF with all the nodes that advertised the new algorithm. The next-hops and RIB entries for the Routing-Plane SIDs MUST be computed from the routing-plane SPF. Certain nodes MAY belong to multiple routing-planes. Such nodes MUST compute SPF corresponding to each plane and compute the next-hops for the SIDs corresponding to each plane.

Each router MAY assign different IP address corresponding to each plane or MAY use the same IP address to assign the node-SIDs and Routing-Plane SIDs. The ingress routers MAY advertise binding-SIDs as defined in [SR-ARCH]section 3.5.2, for the label stacks that are constructed using routing-Plane SIDs. The ingress routers MAY map the incoming IP traffic onto the Routing-Plane SIDs, the mechanisms to do so is implementation dependant and outside the scope of this document.

When the network topology is organized into multiple IGP levels or areas, the Routing Plane SIDs MAY be re-originated from one IGP domain into the other domain by the border routers. The border IGP routers MUST re-advertise the Routing-Plane SIDs if they belong to the corresponding Routing plane and has advertised the algorithm corresponding to the routing-plane.

3.2. Multi-topology SID

Multi topology Routing defines mechanisms to support multiple topologies in a single physical network. ISIS and OSPF extensions to support multi-topology routing is defined in [RFC5120] and [RFC4915] respectively. Multi-topology routing defined in [RFC5120] and [RFC4915] describes mechanisms to separate topologies in ISIS and OSPF by advertising separate MT-TLVs in ISIS and multi-topology scoped Router LSA in OSPF. The different routing planes in customer network can be assigned with different MT-ID for each routing-plane and the multi-topology SIDs can be advertised for each MT-ID as described in [SR-OSPF] and [SR-IS-IS]. Multi-topology SIDs are associated with algorithm 0 and no new algorithm definition is required. All the nodes in the network MUST also support multi-topology routing as defined in [RFC5120] and [RFC4915]. All the nodes in the network compute separate SPF per MT-ID and program the

forwarding planes with MT-SIDs accordingly. Multi-topology SIDs are used to build the explicit paths through the network. Multi-topology based solution has an advantage of possibility of assigning different IGP costs to links for different MTs. For deployments that do not need separate IGP costs and topologies for each routing plane, it comes with an additional operational overhead of maintaining multi-topology configurations.

4. Backward compatibility

The mechanism described in the document is fully backward compatible. If a node does not support the extensions defined in this document, it will not advertise the additional algorithm values in the algorithm sub-TLV. All the computing nodes will not consider the node in the SPF computation if it has not advertised the specific algorithm. For the multi-topology based solution backward compatibility mechanism described in [RFC5120] and [RFC4915] are applicable.

5. Security Considerations

This document does not introduce any further security issues other than those discussed in [SR-OSPF] and [SR-IS-IS].

6. IANA Considerations

This specification updates OSPF and ISIS registry:

OSPF Router Information (RI) TLVs Registry

8 (IANA Preallocated) - SR-Algorithm TLV

Algorithm 2 -5 : SPF in routing plane

ISIS Sub TLVs for Type 242

Type: TBD (suggested value 19)

Description: Segment Routing Algorithm

Algorithm 2-5 : SPF in Routing Plane

7. Acknowledgements

8. References

8.1. 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>.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, DOI 10.17487/RFC4915, June 2007, <http://www.rfc-editor.org/info/rfc4915>.

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <http://www.rfc-editor.org/info/rfc5120>.

[SR-IS-IS] "IS-IS Extensions for Segment Routing, draft-ietf-isis-segment-routing-extensions-11(work in progress)", October 2016.

[SR-OSPF] "OSPF Extensions for Segment Routing, draft-ietf-ospf-segment-routing-extensions-11(work in progress)", July 2016.

[SR-OSPFv3] "OSPFv3 Extensions for Segment Routing, draft-ietf-ospf-ospfv3-segment-routing-extensions-06(work in progress)", July 2016.

8.2. Informative References

[RFC7684] Psenak, P., Gredler, H., Shakir, R., Henderickx, W., Tantsura, J., and A. Lindem, "OSPFv2 Prefix/Link Attribute Advertisement", RFC 7684, DOI 10.17487/RFC7684, November 2015, <http://www.rfc-editor.org/info/rfc7684>.

[SR-ARCH] "Segment Routing Architecture, draft-ietf-spring-segment-routing-09(work in progress)", July 2016.

Authors' Addresses

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

Email: shraddha@juniper.net

Arkadiy Gulko
Thomson Reuters

Email: arkadiy.gulko@thomsonreuters.com

SPRING Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 31, 2017

S. Litkowski
Orange
M. Aissaoui
Nokia

January 27, 2017

Implementing non protected paths using SPRING
draft-litkowski-spring-non-protected-paths-01

Abstract

Segment Routing (SR) leverages the source routing paradigm. A node can steer a packet on a specific path by prepending the packet with an SR header. In the framework of traffic-engineering use cases, a customer may request its service provider to implement some non protected paths. This means that in case of a failure within the network, fast-reroute (or similar) techniques should not be activated for those paths. This document analyzes the different options to implement a non protected path with Segment Routing and in a future release will provide a recommandation on the best option.

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

Table of Contents

1. Problem statement

In some cases, a customer may prefer to react on network failures using its own mechanism. In such cases, the customer usually has two disjoint paths, so a path can take over the traffic in case of failure of the other. The disjoint paths can be provided by a single provider or by multihoming to different providers as displayed in the figure below.

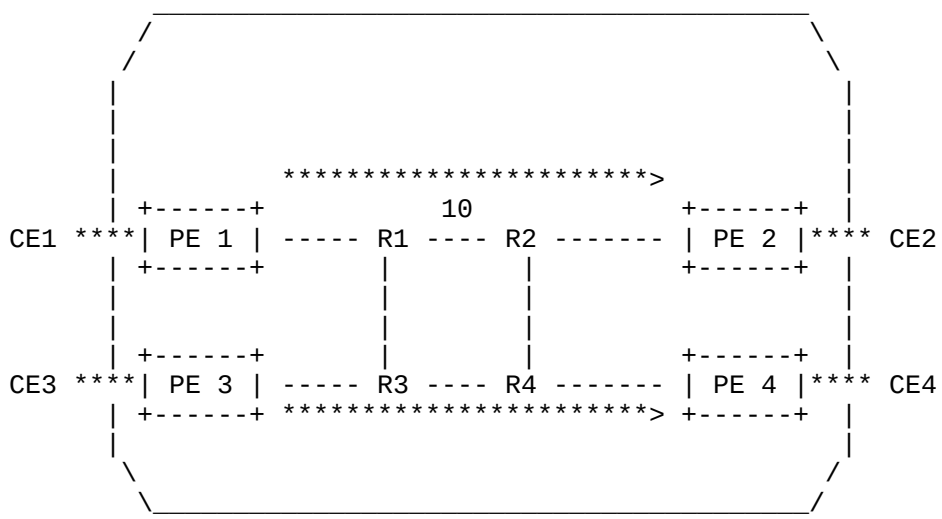

Figure 1 - Disjoint paths provided by a single provider

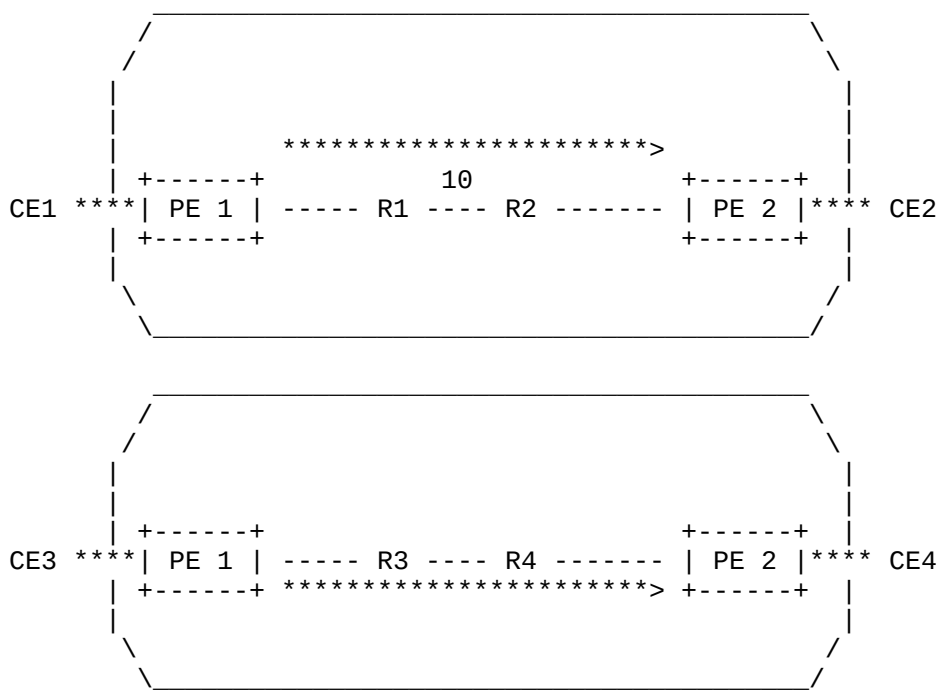

Figure 2 - Disjoint paths provided by using two providers

As the traffic protection is ensured by an end-to-end mechanism at the customer level, the customer requests the service provider to not protect the paths. This is particularly required to avoid both protection mechanisms (customer level and provider level) to be activated at the same time which may lead to unpredictable side effects. However the service provider is allowed to restore the end-to-end path automatically when the primary path is failing by computing and installing a new primary path at the head-end. How the end-to-end protection is handled is out of scope of this document and will be under the customer responsibility.

Another use case could be a service provider selling the traffic protection as a service option. So by default, the provided IP/MPLS path is not protected by any fast-reroute mechanism but the customer can subscribe to an option to activate fast-reroute for its traffic. In the figure 3, the Customer1 service between PE1 and PE2 is protected, in case of failure between R1 and R2, the LSP can use a bypass through R3-R4 nodes until the convergence occurs. The Customer2 did not subscribe to the traffic protection option. If

R3-R4 fails, the traffic between CE3 and CE4 will be disrupted until the convergence occurs.

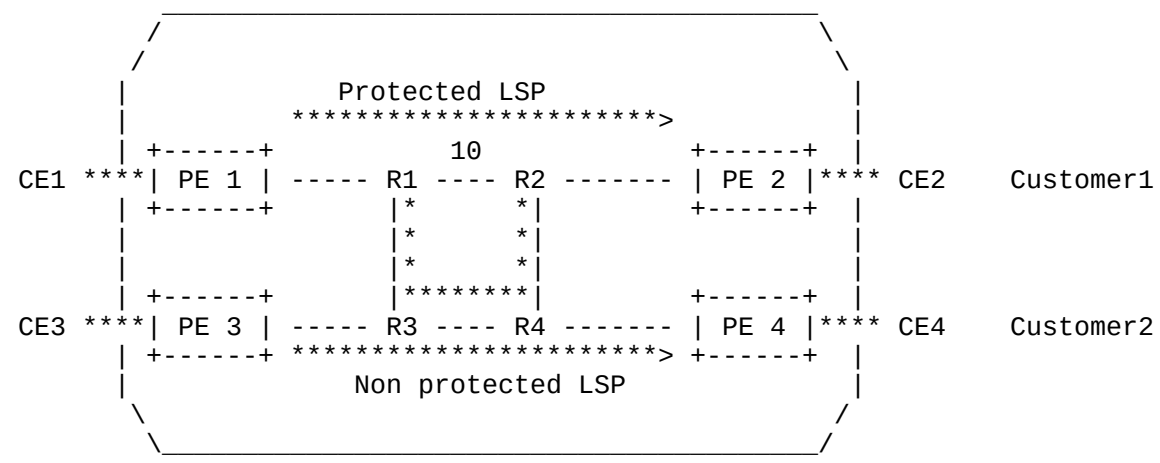

Figure 3 - Provider selling traffic protection as an option

A service provider may also propose a traffic protection service based on path protection rather than local repair on each transit node. In the figure 4, on PE1, two LSPs were created to ensure the customer traffic protection between PE1 and PE2. The primary LSP is used to carry the traffic in the nominal situation. The protection LSP is built as disjoint from the primary LSP and may be preestablished (from controlplane and/or dataplane point of view). When the primary LSP fails, PE1 is responsible to switch the traffic to the protection LSP. As the protection is provided by PE1, both primary and protection LSPs should be setup as non protected so transit nodes will not activate any local-repair mechanism for those LSPs.

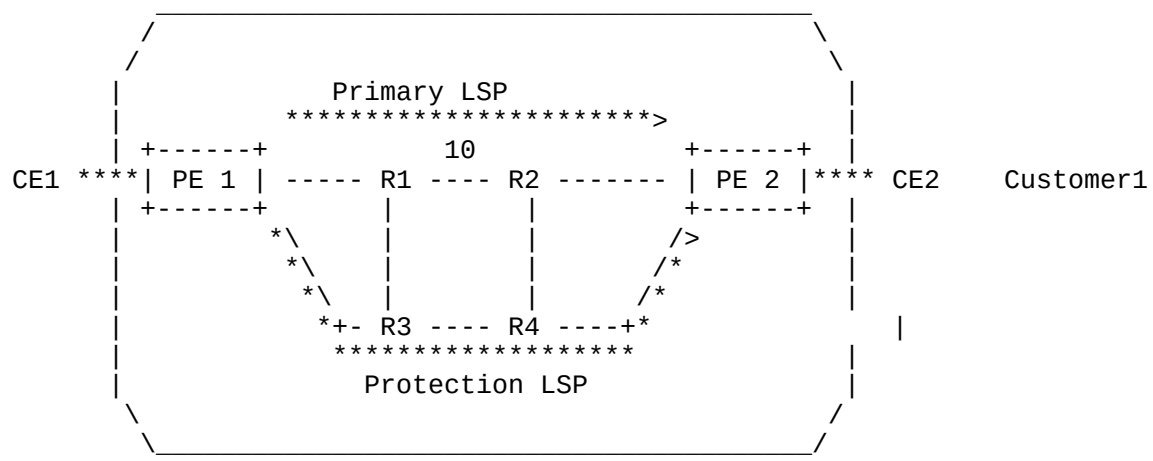

Figure 4 - Provider selling traffic protection as an option

A segment-routing path is expressed as a list of segment identifiers (SID) from different types (Node-SID, Adj-SID, Binding-SID ...). In order to ensure that the segment routing path is not protected, we need to ensure that it does not contain any segment representing a protected path. As an example, in the Figure 1, we consider a path from PE1 to PE2 expressed with the following segment list: {Adj_R1R3,Node_R2,Adj_R2PE2}. If we want to ensure that this path is not protected, we need to ensure that the segment represented by Adj_R1R3 represents a non protected segment, as well as the segments Node_R2 and Adj_R2PE2.

The segment routing path may be computed by a Path Computation Element (PCE). In order to fulfil the non protected path constraint, the PCE needs to be aware of the available SIDs in the network and their protection status.

Several techniques may be used to represent a non protected path with a segment identifier. We propose to analyze the different options.

2. Requirements for a non protected LSP

- o A non protected LSP SHOULD follow a primary path defined based on the constraints of the LSP. This path can be the shortest path (as per the IGP metric) or a more constrained path (explicit path) to fulfil for example a bandwidth, latency or disjointness requirement.

- Upon a failure, a non protected LSP SHOULD be reestablished over a new suitable non-protected path that still fulfils the constraints of the LSP.

- Upon a failure (link, node, srlg...), the traffic of a non protected LSP MUST NOT use any local-repair or any local-rerouting mechanism on transit nodes.

- The computation of a new primary path for the LSP will be handled by the computation node responsible of this LSP (it could be the head-end or a PCE).

- Upon any other traffic-engineering topology change (metric change, overload status change, bandwidth change, latency change...), the non protected LSP MAY be reoptimized to a better path.

2.1. ECMP considerations

When equal cost paths are available within the end-to-end path, implementations may reuse a fast-reroute like mechanism in the dataplane, so when one of the outgoing interface fails, the dataplane switches traffic immediately to the remaining outgoing interfaces in the ECMP set. This behavior is usually hardcoded and cannot be disabled. Based on this assumption, a non protected LSP SHOULD avoid ECMPs.

3. Options to create a non protected path with Segment Routing

3.1. Using only non protected adjacency segments

A node can advertise multiple adjacency segments for a particular link with different properties. The non-protected property is already defined as part of the protocol encodings ([I-D.ietf-isis-segment-routing-extensions], [I-D.ietf-ospf-segment-routing-extensions] and [I-D.gredler-idr-bgp-ls-segment-routing-extension]) through the B flag. However, from an implementation perspective, advertising a protected adjacency segment, a non protected adjacency segment or both for each link is optional.

It is important to note that even if an adjacency segment has the B flag set (protected), it remains up to a local policy of the advertising router to implement the protection or not.

If both protected and non protected Adj-SID are advertised, every node in the network (including PCEs) can be aware of the adjacency segments protection property. When a non protected path is

requested, the path computation module can choose to encode the path with a list a non protected adjacency segments only.

One of the advantage of using only adjacency segments is the insurance that the traffic will never go transiently outside the path defined by the computation module responsible of the path. This solution is fully compliant with the requirements sets in Section 2.

One of the drawbacks of using only adjacency segments is the resulting label stack depth as each hop should require a segment in the stack: crossing 15 nodes, means stacking 15 labels to encode the SR tunnel. Having such a deep stack may be a problem for current hardwares and softwares for either pushing the stack (because the head end is limited in the number of labels it can push) or loadbalancing flows on transit nodes (as deep packet inspection or entropy label look up may be difficult with a deep label stack). Another drawback of advertising both protected and non protected adjacency segments is the additional controlplane and dataplane resource consumption used in the network. As the adjacency SIDs have a local significance, this resource consumption can be considered as negligeable from a data plane point of view. From a control plane point of view, this can also be considered as negligeable with the current CPU and memory usually available on routers.

3.2. Using a combination of node segments and adjacency segments

Using a combination of node segments and adjacency segments is the usual way of creating a segment routing path. However the well known Node-SID (algorithm type Shortest Path) may be protected by a local-repair mechanism by any transit node or may use ECMPs which may be a problem when used for a non protected path. Protecting a particular Node-SID is a matter of a local policy configuration on every node. The following discusses a number of possible approaches.

3.2.1. Adding a protection flag in the Node SID

As for adjacency segments, a new flag may be added in the Prefix-SID to encode the willingness of protection. Each node will then advertise two Node-SIDs (using SPF algorithm), one with the protection flag set, the other without the protection flag set. The same discussion regarding ECMP is also applicable here.

The remaining flag space in the Prefix-SID is small, so adding a new flag requires analysis but this should not be considered as a showstopper.

3.2.2. Using Strict SPF Node SID

[I-D.ietf-spring-segment-routing] defines a Strict Shortest Path algorithm which mandates that the packet is forwarded according to ECMP-aware SPF algorithm and instructs any router in the path to ignore any possible local policy overriding SPF decision. The use of a local-repair for a strict SPF Node-SID is allowed as long as the FRR mechanism enforces the post convergence path to the destination.

This solution does not bring any benefit compared to the regular Node-SID (as it has similar properties).

3.2.3. Using two Node-SIDs with different local policies

Having two instances of the Node-SID (protected and not protected) is a requirement when using Node-SID in protected and non protected paths. The protection of a Node-SID is a matter of a local policy configuration on every node in the network. A service provider may configure two Node-SIDs per node and may adjust the local-repair on every node to protect one Node-SID but not the other. As the protection of the Node-SID is inherited from the protection of the associated prefix, the service provider will need to deploy a new set of prefixes to all nodes to deploy the new set of Node-SIDs. Then it will need to maintain the local-repair policy on every node to ensure that the prefixes associated to the non protected Node-SID are not using the local-repair.

The path computation engine (head-end or PCE) must be aware of the policy defined by the service provider so it can select the right SIDs/prefixes when computing a path.

3.2.4. Advantages and drawbacks

One advantage of combining adjacency and node segments is the reduction of the label stack size.

The drawbacks are the increase of the controlplane and dataplane resource consumption. Whereas having two adjacency SIDs introduces a negligeable impact, having two nodes SIDs increases controlplane and dataplane processing as each node in the network will have to install an MPLS->MPLS and IP->MPLS entry for each additional Node-SID. The regular IP convergence time of the network may be doubled in the worst case while the newly deployed node-SIDs are only used for traffic-engineering applications. One of the other drawback is that a Node-SID may be transiently rerouted on a path that does not fit the constraints anymore if a transit node converges faster than the head-end: this concern is not new and applies to all traffic-engineering use cases. Note that there is a high chance for a

transit node to reroute faster than the head-end as it has usually less computations to run (SPF+CSPFs) and less prefixes to rewrite; it may also run less features leaving more CPU slots for IGP reconvergence. The transient rerouting of the Node-SID may lead to microloops in the network that may impact the customer traffic. Node-SIDs are subject to ECMP and a local-repair mechanism may be implemented for equal cost paths with no way to disable it. If the requirement of preventing any local-repair or ECMP is strict, the path computation engine needs to prevent the usage of all Node-SIDs or needs to detect that a particular Node-SID will be subject to ECMP and enforce the usage of additional adjacency SIDs to break the ECMP. In any case, more adjacency-SIDs will be required in the stack to avoid the ECMP, leading to a deeper label stack.

3.3. Using a combination of adjacency segments and binding-SID

[I-D.ietf-spring-segment-routing] defines the binding segment with multiple use cases. One of the use case of the binding segment is to advertise a tunnel as a segment. When a computation engine computes a non protected path and if the resulting label stack using only non protected adjacency segments is too deep for the network, an external component may create shortcuts in the network by creating a binding segment representing a list of non protected adjacency segments.

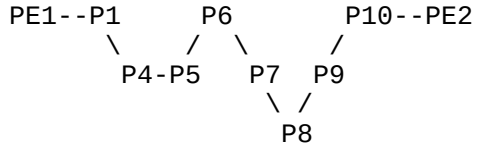

Figure 3 - Use of Binding SID

In the example above, the path from PE1 to PE2 must be expressed with the stack: {Adj_P1P4,Adj_P4P5,Adj_P5P6,Adj_P6P7,Adj_P7P8,Adj_P8P9,Adj_P9P10,Adj_P10PE2}. This stack is too deep due to the limitations of the network. An external component may create a binding Binding1 on P5 that represents the non protected path (P5->P6->P7->P8->P9->P10). When the binding is created and advertised in the topology, the computation engine can use this binding SID in a path, resulting for a PE1 to PE2 path to the stack: {Adj_P1P4,Adj_P4P5,Binding1,Adj_P10PE2}. The usage of the binding SID in the stack allowed to reduce its size to an acceptable value.

One advantage of combining adjacency and binding segments is the reduction of the label stack size. The label stack size can be

reduced to a small amount of labels at some price (creating some states on transit nodes).

The drawbacks are the increase of the controlplane and dataplane resource consumption. This controlplane and dataplane resource consumption are variable and will be linked to the intelligence of the external controller and computation engines and especially how the placement of the bindings is done to maximize the sharing between LSPs. Moreover any optimization try in the binding segment may introduce churn in the network controlplane (Make Before Break can be used to ensure that dataplane is not affected). Programming a binding-SID on a transit node is feasible only if the programming node has the necessary protocol sessions to do so. When a head-end router is performing a path computation, it is usually not the case. When a controller (PCE) is used, it may not have a session to all LSRs in the network, as only edge nodes may require a path computation. The controller may be limited for the placement of the binding SID to the nodes it has a protocol session with (it cannot setup a PCEP session by itself). A full deployment of protocol sessions with the controller may not be feasible for technical reasons (scaling, ...) or economical reasons. A potential mitigation could be to allow protocol sessions to be setup dynamically (when requirement comes) to an authorized subset of nodes in the network: some protocol modifications may be necessary to allow this behavior.

4. Comparison

The following table tries to summarize the various solution pros/cons within a comparison table:

- Solution 1: using adjacency-SIDs only
- Solution 2: using adjacency-SIDs + Node-SIDs with strict SPF algorithm
- Solution 3: using adjacency-SIDs + Node-SIDs with new protection flag
- Solution 4: using adjacency-SIDs + two regular Node-SIDs with a different policy
- Solution 5: using adjacency-SIDs + Binding-SIDs

We consider a network with N nodes and L links, with an average of l links per node.

Criteria	Soluti	Solution	Solution 3	Solution 4	Solutio

	on 1	2			n 5
Label stack size	One label per hop	Reduced	Reduced	Reduced	Reduced
Controlplane	Negligible	Potential additional computation + 2*N entries in RIB	+ 2*N entries in RIB	+ 2*N entries in RIB	Adds states in the LSRs
Dataplane	+l entries	+2*N entries	+2*N entries	+2*N entries	Variable
IP convergence time	None	Double	Double	Double	None
Computation engine	Needs to select Adj-SIDs with B=0	Needs to select Adj-SIDs with B=0 and Node-SIDs with strict SPF	Needs to select Adj-SIDs with B=0 and Node-SIDs with B=0	Needs to select Adj-SIDs with B=0 and needs to understand policy from the SP to select the right Node-SIDs	Needs to select Adj-SIDs with B=0 and place the binding SID in a smart way
Protocol	None	None	Need a new flag	None	None
ECMP avoidance	Supported	Supported at the price of increasing the label stack	Supported at the price of increasing the label stack	Supported at the price of increasing the label stack	Supported
Requirem	Yes	Partially	Partially	Partially	Yes

ents ful filment		(allows ECMP+transient rerouting)	(allows ECMP+transient rerouting)	(allows ECMP+transient rerouting)	
Others	None	None	None	None	Requires a controller with sessions to all nodes (even transit)

Comparison of solutions

5. Recommended option(s)

Based on the analysis in Section 4, we only have two solutions that fulfill the requirements expressed in Section 2: usage of adjacency-SIDs only, usage of a combination of adjacency SIDs and binding SIDs.

As using only Adjacency-SIDs may reduce today the possibility of creating a path (due to the hardware/software limitations), authors would like to encourage the usage of a combination of adjacency-SIDs and binding-SIDs (Section 3.3) as a short-term solution.

However this approach has also several drawbacks, but authors think that these drawbacks can be reduced by enhancing existing protocols.

As a long term solution, authors would like to encourage vendors to support the ability for a node to push a significant number of labels, up to the full network diameter.

6. Security Considerations

TBD.

7. Acknowledgements

Authors would like to thank Bruno Decraene for his valuable comments.

8. IANA Considerations

N/A

9. Normative References

[I-D.gredler-idr-bgp-ls-segment-routing-extension]
Gredler, H., Ray, S., Previdi, S., Filsfils, C., Chen, M., and J. Tantsura, "BGP Link-State extensions for Segment Routing", draft-gredler-idr-bgp-ls-segment-routing-extension-02 (work in progress), October 2014.

[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-09 (work in progress), October 2016.

[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-10 (work in progress), October 2016.

[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-10 (work in progress), November 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>.

Authors' Addresses

Stephane Litkowski
Orange

Email: stephane.litkowski@orange.com

Mustapha Aissaoui
Nokia

Email: mustapha.aissaoui@nokia.com