SPRING
Internet-Draft
Intended status: Standards Track
Expires: July 12, 2018

F. Clad, Ed.
Cisco Systems, Inc.
X. Xu, Ed.
Huawei
C. Filsfils
Cisco Systems, Inc.
D. Bernier
Bell Canada
B. Decraene
Orange
S. Ma
Juniper
C. Yadlapalli
AT&T
W. Henderickx
Nokia
S. Salsano
Universita di Roma "Tor Vergata"
January 8, 2018

Segment Routing for Service Chaining
draft-xuclad-spring-sr-service-chaining-00

Abstract

This document defines data plane functionality required to implement service segments and achieve service chaining in SR-enabled MPLS and IP networks, as described in the Segment Routing architecture.

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 https://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 July 12, 2018.

Table of Contents

1. Introduction

Segment Routing (SR) is an architecture based on the source routing paradigm that seeks the right balance between distributed intelligence and centralized programmability. SR can be used with an MPLS or an IPv6 data plane to steer packets through an ordered list of instructions, called segments. These segments may encode simple routing instructions for forwarding packets along a specific network path, or rich behaviors to support use-cases such as Service Function Chaining (SFC).

In the context of SFC, each Service Function (SF), running either on a physical appliance or in a virtual environment, is associated with a segment, which can then be used in a segment list to steer packets through the SF. Such service segments may be combined together in a segment list to achieve SFC, but also with other types of segments as defined in [I-D.ietf-spring-segment-routing]. SR thus provides a fully integrated solution for SFC, overlay and underlay optimization. Furthermore, the IPv6 dataplane natively supports metadata transportation as part of the SR information attached to the packets.

This document describes how SR enables SFC in a simple and scalable manner, from the segment association to the SF up to the traffic classification and steering into the service chain. Several SR proxy behaviors are also defined to support SR SFC through legacy, SR-unaware, SFs in various circumstances.

The definition of an SR Policy and the steering of traffic into an SR Policy is outside the scope of this document. These aspects are covered in [I-D.filsfils-spring-segment-routing-policy].

The definition of control plane components, such as service segment discovery, is outside the scope of this data plane document. BGP

extensions to support SR-based SFC are proposed in [I-D.dawra-idr-bgp-sr-service-chaining].

Familiarity with the following IETF documents is assumed:

- Segment Routing Architecture [I-D.ietf-spring-segment-routing]
- Segment Routing with MPLS data plane [I-D.ietf-spring-segment-routing-mpls]
- Segment Routing Traffic Engineering Policy [I-D.filsfils-spring-segment-routing-policy]
- Segment Routing Header [I-D.ietf-6man-segment-routing-header]
- SRv6 Network Programming [I-D.filsfils-spring-srv6-network-programming]
- Unified Source Routing Instruction using MPLS Label Stack [I-D.xu-mpls-unified-source-routing-instruction]
- Service Function Chaining Architecture [RFC7665]

2. Terminology

This document leverages the terminology introduced in [I-D.ietf-spring-segment-routing], [I-D.filsfils-spring-segment-routing-policy] and [RFC7665]. It also introduces the following new terminology.

SR-aware SF: Service Function fully capable of processing SR traffic

SR-unaware SF: Service Function unable to process SR traffic or behaving incorrectly for such traffic

SR proxy: Proxy handling the SR processing on behalf of an SR-unaware SF

Service Segment: Segment associated with an SF, either directly or via an SR proxy

SR SFC policy: SR policy, as defined in [I-D.filsfils-spring-segment-routing-policy], that includes at least one Service Segment. An SR SFC policy may also contain other types of segments, such as VPN or TE segments.

3. Classification and steering

Classification and steering mechanisms are defined in section 12 of [I-D.filsfils-spring-segment-routing-policy] and are independent from the purpose of the SR policy. From a headend perspective, there is no difference whether a policy contains IGP, BGP, peering, VPN and service segments, or any combination of these.

As documented in the above reference, traffic is classified when entering an SR domain. The SR policy head-end may, depending on its capabilities, classify the packets on a per-destination basis, via simple FIB entries, or apply more complex policy routing rules requiring to look deeper into the packet. These rules are expected to support basic policy routing such as 5-tuple matching. In addition, the IPv6 SRH tag field defined in [I-D.ietf-6man-segment-routing-header] can be used to identify and classify packets sharing the same set of properties. Classified traffic is then steered into the appropriate SR policy, which is associated with a weighted set of segment lists.

SR traffic can be re-classified by an SR endpoint along the original SR policy (e.g., DPI service) or a transit node intercepting the traffic. This node is the head-end of a new SR policy that is imposed onto the packet, either as a stack of MPLS labels or as an IPv6 and SRH encapsulation.

4. Service Functions

A Service Function (SF) may be a physical appliance running on dedicated hardware, a virtualized service inside an isolated environment such as a VM, container or namespace, or any process running on a compute element. An SF may also comprise multiple sub-components running in different processes or containers. Unless otherwise stated, this document does not make any assumption on the type or execution environment of an SF.

SR enables SFC by assigning a segment identifier, or SID, to each SF and sequencing these service SIDs in a segment list. A service SID may be of local significance or directly reachable from anywhere in the routing domain. The latter is realized with SR-MPLS by assigning a SID from the global label block ([I-D.ietf-spring-segment-routing-mpls]), or with SRv6 by advertising the SID locator in the routing protocol ([I-D.filsfils-spring-srv6-network-programming]). It is up to the network operator to define the scope and reachability of each service SID. This decision can be based on various considerations such as infrastructure dynamicity, available control plane or orchestration system capabilities.

This document categorizes SFs in two types, depending on whether they are able to behave properly in the presence of SR information or not. These are respectively named SR-aware and SR-unaware SFs. An SR-aware SF can process the SR information in the packets it receives. This means being able to identify the active segment as a local instruction and move forward in the segment list, but also that the SF own behavior is not hindered due to the presence of SR information. For example, an SR-aware firewall filtering SRv6 traffic based on its final destination must retrieve that information from the last entry in the SRH rather than the Destination Address field of the IPv6 header. Any SF that does not meet these criteria is considered as SR-unaware.

4.1. SR-aware SFs

An SR-aware SF is associated with a locally instantiated service segment, which is used to steer traffic through it.

If the SF is configured to intercept all the packets passing through the appliance, the underlying routing system only has to implement a default SR endpoint behavior (SR-MPLS node segment or SRv6 End function), and the corresponding SID will be used to steer traffic through the SF.

If the SF requires the packets to be directed to a specific virtual interface, networking queue or process, a dedicated SR behavior may be required to steer the packets to the appropriate location. The definition of such SF-specific functions is out of the scope of this document.

An SRv6-aware SF may also retrieve, store or modify information in the SRH TLVs.

4.2. SR-unaware SFs

An SR-unaware SF is not able to process the SR information in the traffic that it receives. It may either drop the traffic or take erroneous decisions due to the unrecognized routing information. In order to include such SFs in an SR SC policy, it is thus required to remove the SR information as well as any other encapsulation header before the SF receives the packet, or to alter it in such a way that the SF can correctly process the packet.

In this document, we define the concept of an SR proxy as an entity, separate from the SF, that performs these modifications and handle the SR processing on behalf of an SF. The SR proxy can run as a separate process on the SF appliance, on a virtual switch or router

on the compute node or on a remote host. In this document, we only assume that the proxy is connected to the SF via a layer-2 link.

An SR-unaware SF is associated with a service segment instantiated on the SR proxy, which is used to steer traffic through the SF. Section 6 describes several SR proxy behaviors to handle the encapsulation headers and SR information under various circumstances.

5. Service function chaining

When applying a particular Service Function Chain (SFC) [RFC7665] to the traffic selected by a service classifier, the traffic need to be steered through an ordered set of Service Functions (SF) in the network. This ordered set of SFs in the network indicates the Service Function Path (SFP) associated with the above SFC. In order to steer the selected traffic through the required ordered list of SFs, the service classifier needs to attach information to the packet specifying exactly which Service Function Forwarders (SFFs) and which SFs are to be visited by traffic, the SFC, or the partially specified SFP which is in between the former two extremes.

The SR source routing mechanisms can be used to steer traffic through an ordered set of devices (i.e., an explicit path) and instruct those nodes to execute specific operations on the packet.

This section describes how to leverage SR to realize a transport-independent service function chaining by encoding the service function path information or service function chain information as an MPLS label stack or an IPv6 SRH.

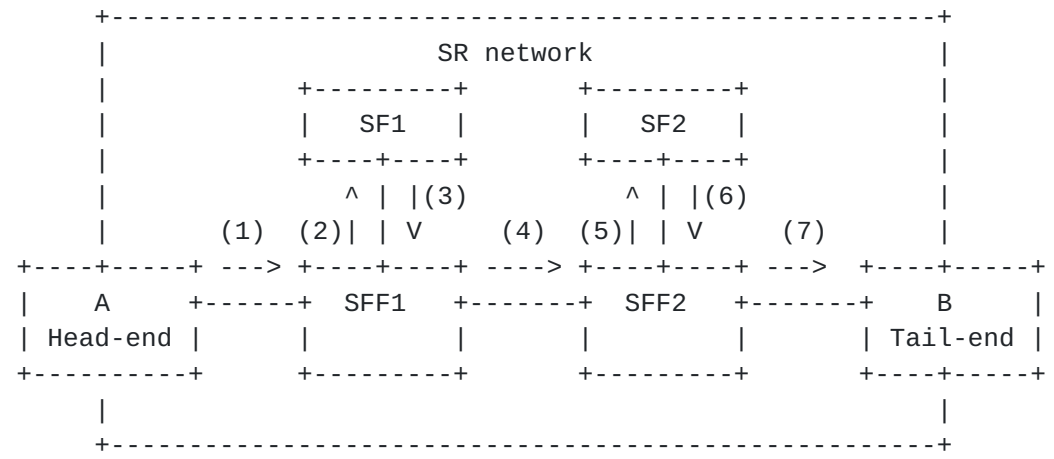

Figure 1: Service Function Chaining in SR networks

As shown in Figure 1, SFF1 and SFF2 are two SR-capable nodes. They are also SFFs, each with one SF attached. In addition, they have

allocated and advertised segments for their locally attached SFs. For example, SFF1 allocates and advertises a SID (i.e., S(SF1)) for SF1 while SFF2 allocates and advertises a SID (i.e., S(SF2)) for SF2. These SIDs, which are used to indicate SFs, are referred to as service segments, while the SFFs are identified by either node or adjacency segments depending on how strictly the network path needs to be specified. In this example, we assume that the traffic is steered to both SFFs using their node segments S(SFF1) and S(SFF2), respectively.

Now assume that a given traffic flow is steered in an SR policy instantiated on node A with an endpoint B, hereafter referred to as the SR policy head-end and tail-end respectively, and associated with particular SFC requirements (i.e., SF1-> SF2). From an SR policy perspective, SFC is only a particular case of traffic engineering where the SR path includes service functions. An SR-SFC policy inherits all the properties of SR-TE policies as defined in [I-D.filsfils-spring-segment-routing-policy]. Section 5.1 and Section 5.2 describe approaches of leveraging the SR-MPLS and SRv6 mechanisms to realize stateless service function chaining. The complete SFP and SFC information is encoded within an MPLS label stack or an IPv6 SRH carried by the packets, so that no per-chain state is required at the intermediate hops. Since the encoding of the partially specified SFP is just a simple combination of the encoding of the SFP and the encoding of the SFC, this document would not describe how to encode the partially specified SFP anymore.

5.1. SR-MPLS data plane

5.1.1. Encoding SFP Information by an MPLS Label Stack

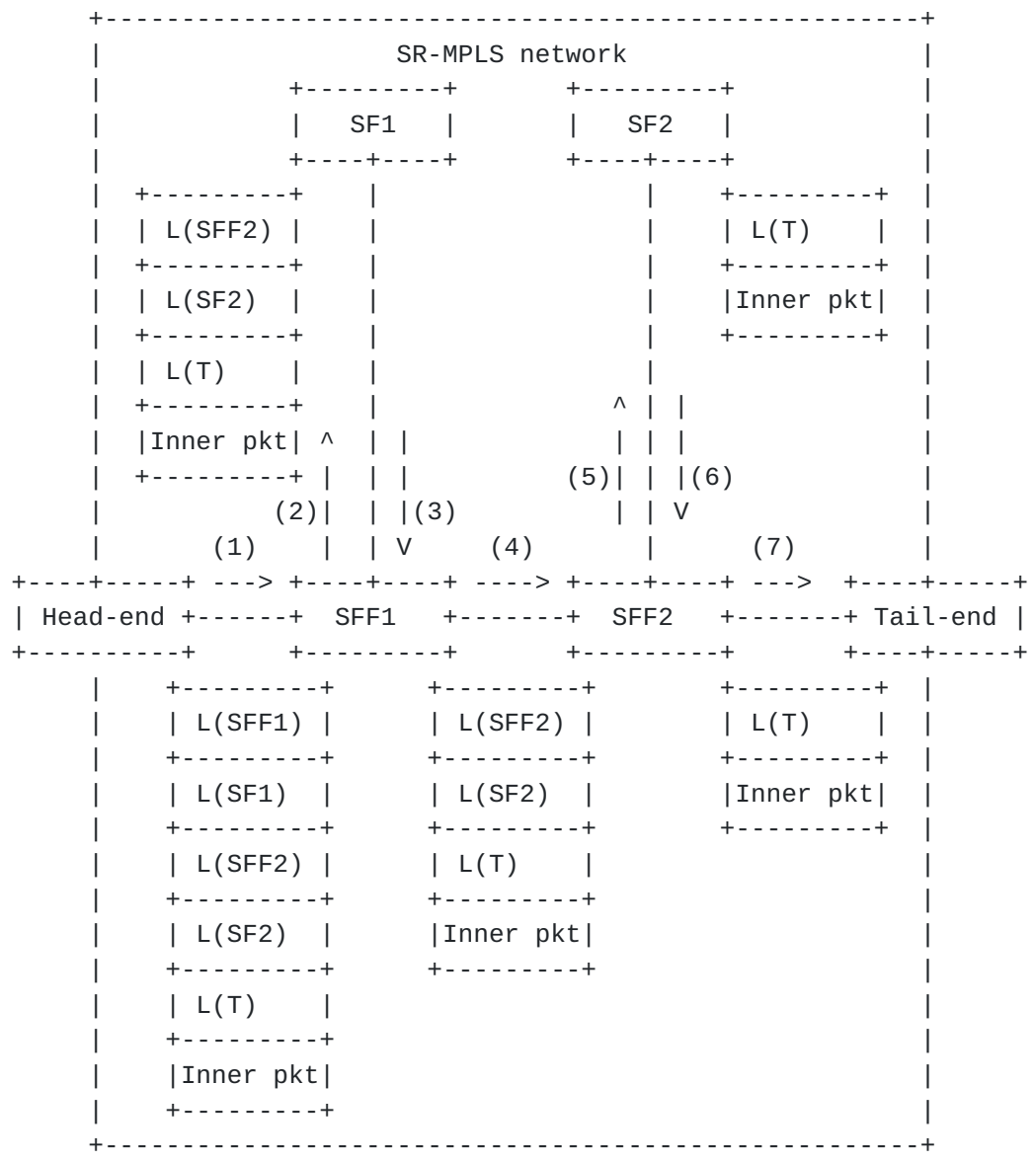

Figure 2: Packet walk in MPLS underlay

As shown in Figure 2, the head-end, acting as a service classifier, determines that the selected packet needs to travel through an SFC (SF1->SF2) and steers this packet into the appropriate SR policy as described in [I-D.filsfils-spring-segment-routing-policy]. As a result, the packet is encapsulated with an MPLS label stack containing the segment list <SFF1, SF1, SFF2, SF2, T>. This segment list encodes in a stateless manner the SFP corresponding to the above SFC as an MPLS label stack where each service segment is a local MPLS label allocated from SFFs' label spaces. To some extent, the MPLS label stack here could be looked as a specific implementation of the SFC encapsulation used for containing the SFP information [RFC7665], which does not require the SFF to maintain per-chain state.

When the encapsulated packet arrives at SFF1, SFF1 knows how to send the packet to SF1 based on the top label (i.e., L(SF1)) of the received MPLS packet. We first consider the case where SF1 is an SR-aware SF, i.e., it understands how to process a packet with a pre-pended SR-MPLS label stack. In this case the packet would be sent to SF1 by SFF1 with the label stack L(SFF2)->L(SF2). SF1 would perform the required service function on the received MPLS packet where the payload type is determined using the first nibble of the MPLS payload. After the MPLS packet is returned from SF1, SFF1 would send it to SFF2 according to the top label (i.e., L(SFF2)).

If SF1 is an SR-unaware SF, i.e. one that is unable to process the MPLS label stack, the remaining MPLS label stack (i.e., L(SFF2)->L(SF2)) MUST be stripped from the packet before sending the packet to SF1. When the packet is returned from SF1, SFF1 would re-impose the MPLS label stack which had been previously stripped and then send the packet to SFF2 according to the current top label (i.e., L(SFF2)). Proxy mechanisms to support SR-unaware SFs are proposed in section 6 of this document.

When the encapsulated packet arrives at SFF2, SFF2 would perform the similar action to that described above.

By leveraging the SR-MPLS data plane, [I-D.xu-mpls-unified-source-routing-instruction] describes a source routing instruction which works across both IPv4 and IPv6 underlays in addition to the MPLS underlay. As shown in Figure 3, if there is no MPLS LSP towards the next node segment (i.e., the next SFF identified by the current top label), the corresponding IP-based tunnel for MPLS (e.g., MPLS-in-IP/GRE tunnel [RFC4023], MPLS-in-UDP tunnel [RFC7510] or MPLS-in-L2TPv3 tunnel [RFC4817]) would be used.

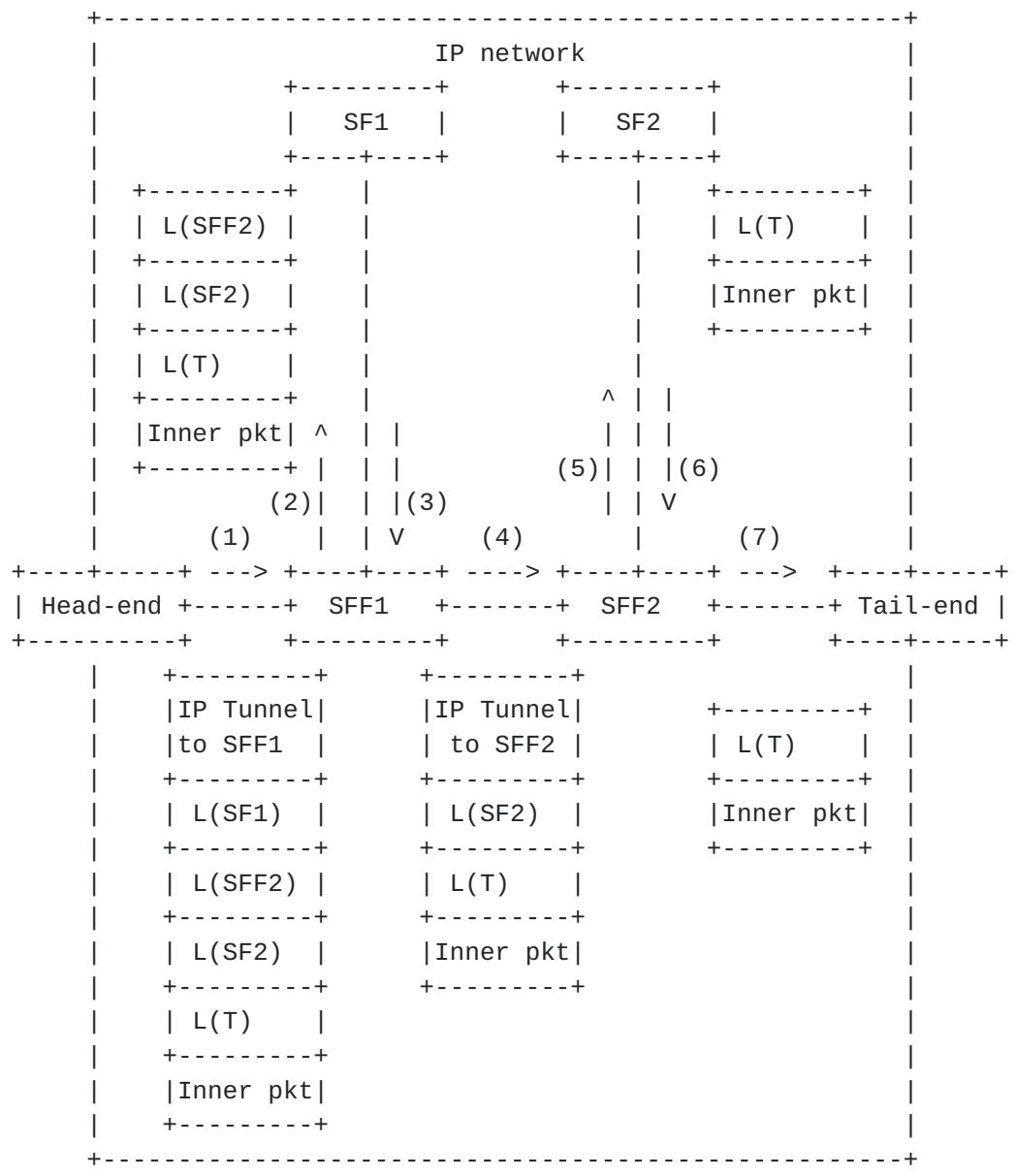

Figure 3: Packet walk in IP underlay

Since the transport (i.e., the underlay) could be IPv4, IPv6 or even MPLS networks, the above approach of encoding the SFP information by an MPLS label stack is fully transport-independent which is one of the major requirements for the SFC encapsulation [RFC7665].

5.1.2. Encoding SFC Information by an MPLS Label Stack

The head-end, acting as a service classifier, determines that the selected packet needs to travel through an SFC (SF1->SF2) and steers this packet into the appropriate SR policy as described in [I-D.filsfils-spring-segment-routing-policy]. This results in the

packet being encapsulated with an MPLS label stack containing the segment list <SF1, SF2, T>, which encodes that SFC. Those SF labels MUST be domain-wide unique MPLS labels. Since it is known to the service classifier that SFF1 is attached with an instance of SF1, the service classifier would therefore send the MPLS encapsulated packet through either an MPLS LSP tunnel or an IP-based tunnel towards SFF1 (as shown in Figure 4 and Figure 5 respectively). When the MPLS encapsulated packet arrives at SFF1, SFF1 would know which SF should be performed according to the current top label (i.e., L(SF1)). Similarly, SFF1 would send the packet returned from SF1 to SFF2 through either an MPLS LSP tunnel or an IP-based tunnel towards SFF2 since it's known to SFF1 that SFF2 is attached with an instance of SF2. When the encapsulated packet arrives at SFF2, SFF2 would do the similar action as what has been done by SFF1. Since the transport (i.e., the underlay) could be IPv4, IPv6 or even MPLS networks, the above approach of encoding the SFC information by an MPLS label stack is fully transport-independent which is one of the major requirements for the SFC encapsulation [RFC7665].

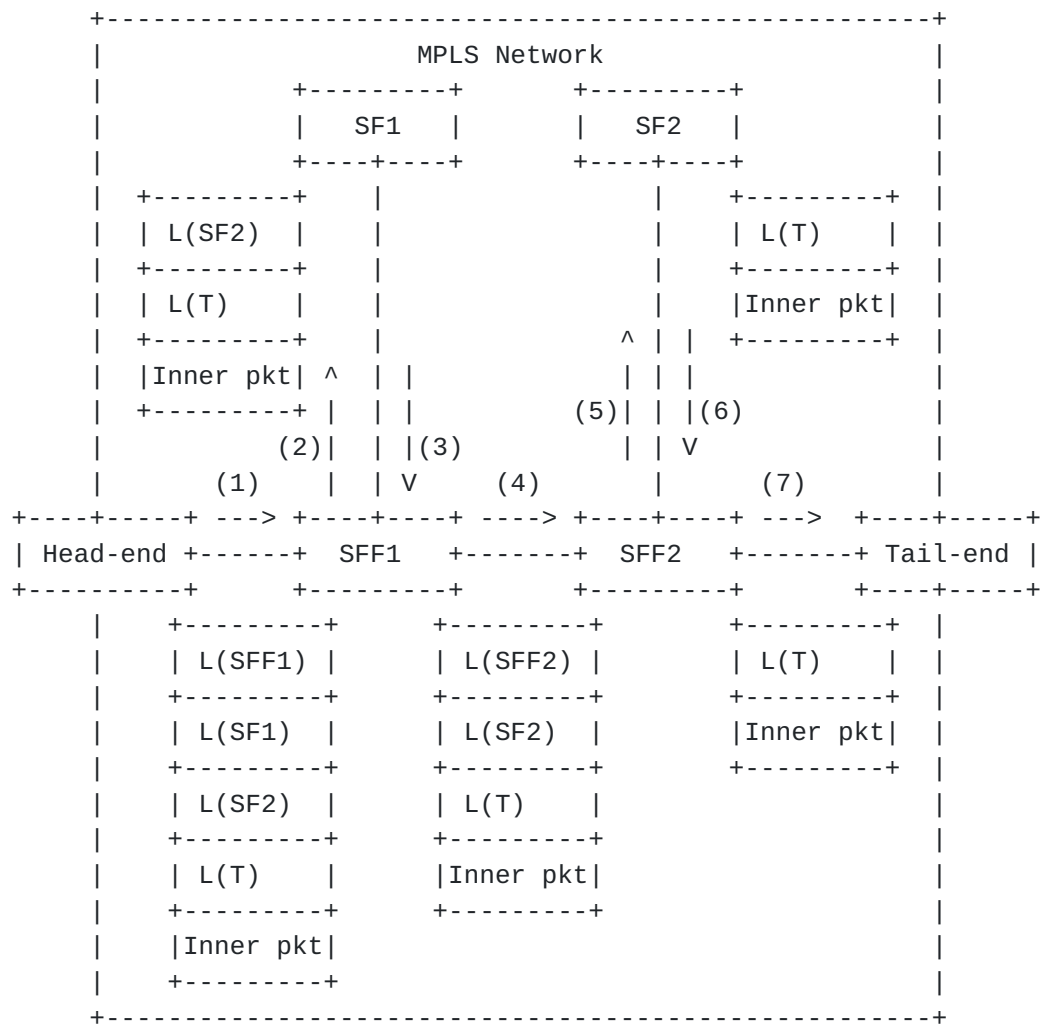

Figure 4: Packet walk in MPLS underlay

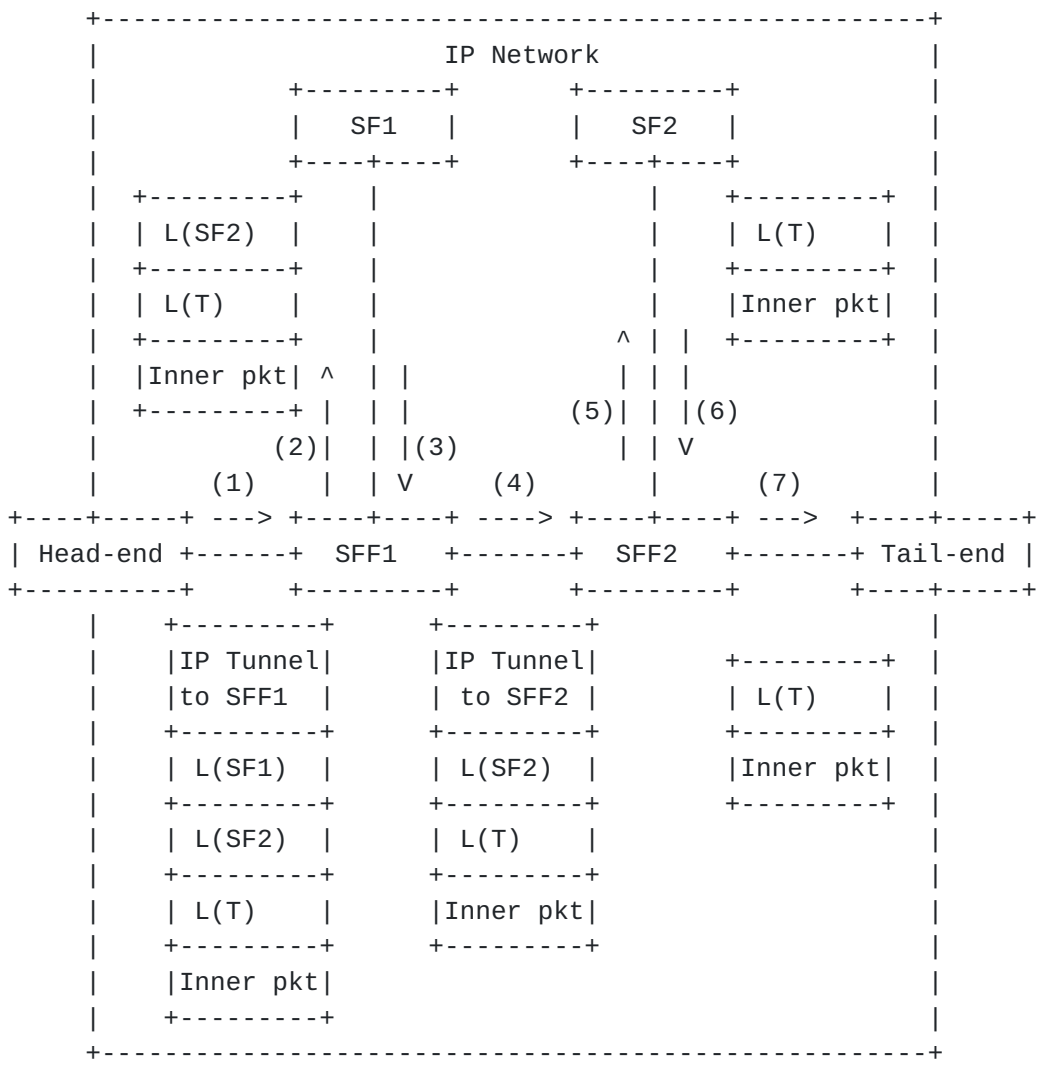

Figure 5: Packet walk in IP underlay

5.2. SRv6 data plane

5.2.1. Encoding SFP Information by an SRv6 SRH

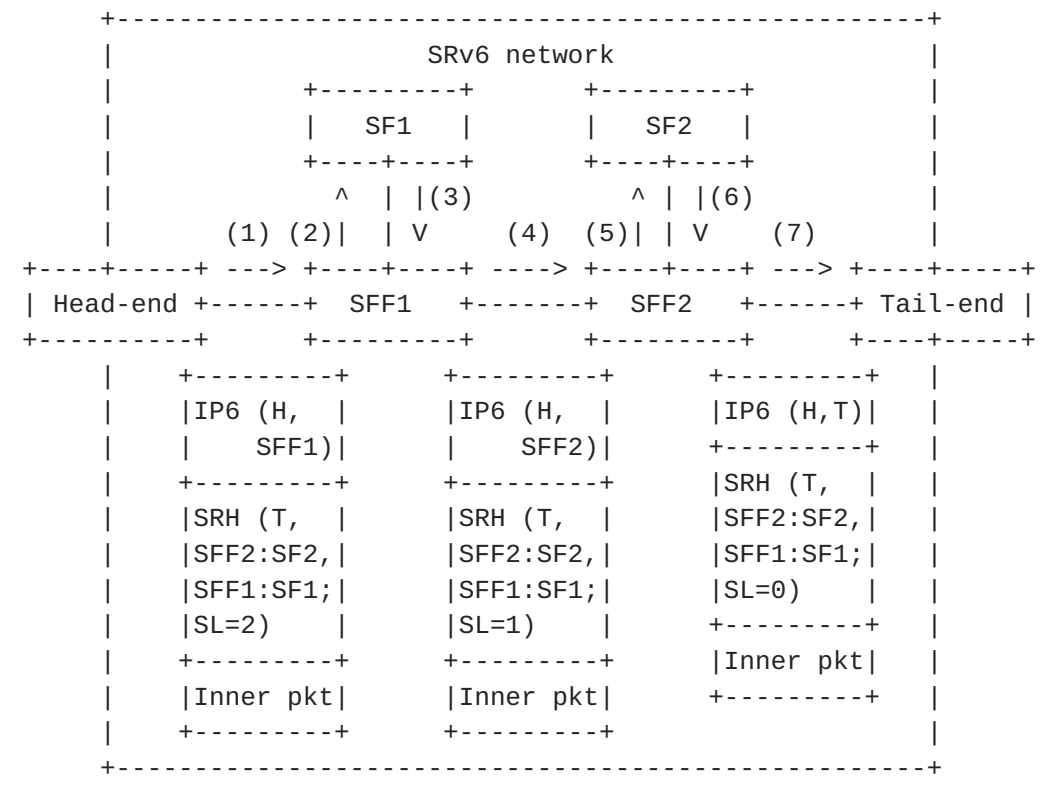

Figure 6: Packet walk in SRv6 network

As shown in Figure 6, the head-end, acting as a service classifier, determines that the selected packet needs to travel through an SFC (SF1->SF2) and steers this packet into the appropriate SR policy as described in [I-D.filsfils-spring-segment-routing-policy]. As a result, the packet is encapsulated with an IPv6 header and an SRH containing the segment list <SFF1:SF1, SFF2:SF2, T>. The intermediate segments in this list leverage the SRv6 locator-function concept introduced in [I-D.filsfils-spring-srv6-network-programming] to encode both the SFF and the SF in a single IPv6 SID. The traffic is steered via regular IPv6 forwarding up to the SFF represented in the locator part of the SID and then passed to the SF identified by the SID function. This SRH thus indicates in a stateless manner the SFP corresponding to the above SFC. To some extent, the SRH here could be looked as a specific implementation of the SFC encapsulation used for containing the SFP information [RFC7665], which does not require the SFF to maintain per-chain state.

When the encapsulated packet arrives at SFF1, SFF1 knows how to send the packet to the SF based on the active segment. We first consider the case where SF1 is an SR-aware SF, i.e., it understands how to process an IPv6 encapsulated packet with an SRH. In this case the packet is sent to SF1 by SFF1 with the IP and SR headers (H,SFF2:SF2)(T,SFF2:SF2,SFF1:SF1;SL=1). SF1 performs the required

service function on the received packet, where the payload is determined based on the Next Header field value of last extension header and/or the active segment. After the packet is returned from SF1, SFF1 simply forwards it to SFF2 according to the IPv6 destination address.

If SF1 is an SR-unaware SF, i.e. one that is unable to process IPv6 encapsulated packets with an SRH, the encapsulation headers (i.e., outer IPv6 with any extension header) MUST be stripped from the packet before it is sent to SF1. When the packet is returned from SF1, SFF1 would re-encapsulate the packet with the IPv6 and SR headers that had been previously stripped and then send the packet to SFF2 according to the IPv6 destination address. Proxy mechanisms to support SR-unaware SFs are proposed in section 6 of this document.

When the encapsulated packet arrives at SFF2, SFF2 would perform the similar action to that described above.

5.2.2. Encoding SFC Information by an IPv6 SRH

The head-end, acting as a service classifier, determines that the selected packet needs to travel through an SFC (SF1->SF2) and steers this packet into the appropriate SR policy. This results in the packet being encapsulated with an IPv6 header and an SRH containing the segment list <A1:SF1, A2:SF2, T>. In this case, the locator parts A1 and A2 of the intermediate service segments are anycast prefixes advertised by several SFFs attached to SF1 and SF2, respectively. The policy head-end may thus let the traffic be steered to the closest instance of each SF or add intermediate segments to select a particular SF instance. Furthermore, since it is known to the head-end that SFF1 is attached to an instance of SF1, the encapsulated packet may be sent to SFF1 through an MPLS LSP or an IP-based tunnel. Similar tunneling can then be performed between SFF1 and SFF1, and between SFF2 and the tail-end, as illustrated on Figure 7. Since the transport (i.e., the underlay) could be IPv4, IPv6 or even MPLS, the above approach of encoding the SFC information by an IPv6 SRH is fully transport-independent which is one of the major requirements for the SFC encapsulation [RFC7665].

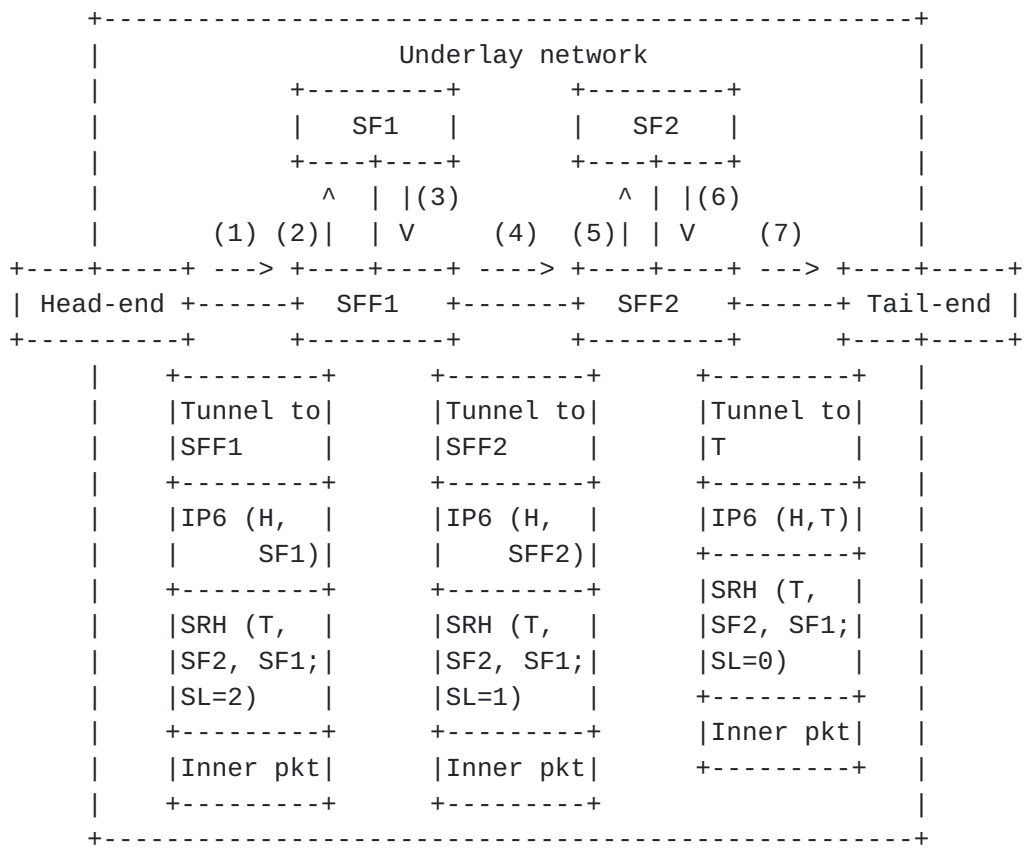

Figure 7: Packet walk in underlay network

6. SR proxy behaviors

This section describes several SR proxy behaviors designed to enable SR SFC through SR-unaware SFs. A system implementing one of these functions may handle the SR processing on behalf of an SR-unaware SF and allows the SF to properly process the traffic that is steered through it.

An SF may be located at any hop in an SR policy, including the last segment. However, the SR proxy behaviors defined in this section are dedicated to supporting SR-unaware SFs at intermediate hops in the segment list. In case an SR-unaware SF is at the last segment, it is sufficient to ensure that the SR information is ignored (IPv6 routing extension header with Segments Left equal to 0) or removed before the packet reaches the SF (MPLS PHP, SRv6 End.D or PSP).

As illustrated on Figure 8, the generic behavior of an SR proxy has two parts. The first part is in charge of passing traffic from the network to the SF. It intercepts the SR traffic destined for the SF via a locally instantiated service segment, modifies it in such a way that it appears as non-SR traffic to the SF, then sends it out on a

given interface, IFACE-OUT, connected to the SF. The second part receives the traffic coming back from the SF on IFACE-IN, restores the SR information and forwards it according to the next segment in the list. Unless otherwise stated IFACE-OUT and IFACE-IN can represent the same interface.

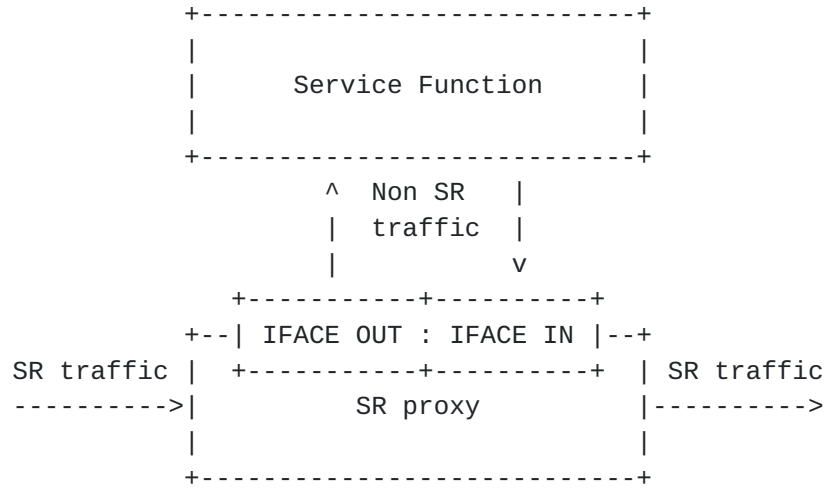

Figure 8: Generic SR proxy

In the next subsections, the following SR proxy mechanisms are defined:

- Static proxy
- Dynamic proxy
- Shared-memory proxy
- Masquerading proxy

Each mechanism has its own characteristics and constraints, which are summarized in the below table. It is up to the operator to select the best one based on the proxy node capabilities, the SF behavior and the traffic type. It is also possible to use different proxy mechanisms within the same service chain.

		Static	Dynamic	Shared memory	Masquerading
SR flavors	SR-MPLS	Y	Y	Y	-
	SRv6 insertion	P	P	P	Y
	SRv6 encapsulation	Y	Y	Y	-
Inner header	Ethernet	Y	Y	Y	-
	IPv4	Y	Y	Y	-
	IPv6	Y	Y	Y	-
Chain agnostic configuration		N	N	Y	Y
Transparent to chain changes		N	Y	Y	Y
SF support	DA modification	Y	Y	Y	NAT
	Payload modification	Y	Y	Y	Y
	Packet generation	Y	Y	cache	cache
	Packet deletion	Y	Y	Y	Y
	Transport endpoint	Y	Y	cache	cache

Figure 9: SR proxy summary

Note: The use of a shared memory proxy requires both the SF and the proxy to be running on the same node.

6.1. Static SR proxy

The static proxy is an SR endpoint behavior for processing SR-MPLS or SRv6 encapsulated traffic on behalf of an SR-unaware SF. This proxy thus receives SR traffic that is formed of an MPLS label stack or an IPv6 header on top of an inner packet, which can be Ethernet, IPv4 or IPv6.

A static SR proxy segment is associated with the following mandatory parameters:

- INNER-TYPE: Inner packet type
- S-ADDR: Ethernet or IP address of the SF (only for inner type IPv4 and IPv6)
- IFACE-OUT: Local interface for sending traffic towards the SF
- IFACE-IN: Local interface receiving the traffic coming back from the SF
- CACHE: SR information to be attached on the traffic coming back from the SF, including at least
  - CACHE.SA: IPv6 source address (SRv6 only)
  - CACHE.LIST: Segment list expressed as MPLS labels or IPv6 address

A static SR proxy segment is thus defined for a specific SF, inner packet type and cached SR information. It is also bound to a pair of directed interfaces on the proxy. These may be both directions of a single interface, or opposite directions of two different interfaces. The latter is recommended in case the SF is to be used as part of a bi-directional SR SC policy. If the proxy and the SF both support 802.1Q, IFACE-OUT and IFACE-IN can also represent sub-interfaces.

The first part of this behavior is triggered when the proxy node receives a packet whose active segment matches a segment associated with the static proxy behavior. It removes the SR information from the packet then sends it on a specific interface towards the associated SF. This SR information corresponds to the full label stack for SR-MPLS or to the encapsulation IPv6 header with any attached extension header in the case of SRv6.

The second part is an inbound policy attached to the proxy interface receiving the traffic returning from the SF, IFACE-IN. This policy attaches to the incoming traffic the cached SR information associated

with the SR proxy segment. If the proxy segment uses the SR-MPLS data plane, CACHE contains a stack of labels to be pushed on top the packets. With the SRv6 data plane, CACHE is defined as a source address, an active segment and an optional SRH (tag, segments left, segment list and metadata). The proxy encapsulates the packets with an IPv6 header that has the source address, the active segment as destination address and the SRH as a routing extension header. After the SR information has been attached, the packets are forwarded according to the active segment, which is represented by the top MPLS label or the IPv6 Destination Address.

In this scenario, there are no restrictions on the operations that can be performed by the SF on the stream of packets. It may operate at all protocol layers, terminate transport layer connections, generate new packets and initiate transport layer connections. This behavior may also be used to integrate an IPv4-only SF into an SRv6 policy. However, a static SR proxy segment can be used in only one service chain at a time. As opposed to most other segment types, a static SR proxy segment is bound to a unique list of segments, which represents a directed SR SC policy. This is due to the cached SR information being defined in the segment configuration. This limitation only prevents multiple segment lists from using the same static SR proxy segment at the same time, but a single segment list can be shared by any number of traffic flows. Besides, since the returning traffic from the SF is re-classified based on the incoming interface, an interface can be used as receiving interface (IFACE-IN) only for a single SR proxy segment at a time. In the case of a bi-directional SR SC policy, a different SR proxy segment and receiving interface are required for the return direction.

6.1.1. SR-MPLS pseudocode

6.1.1.1. Static proxy for inner type Ethernet

Upon receiving an MPLS packet with top label L, where L is an MPLS L2 static proxy segment, a node N does:

```
1.   IF payload type is Ethernet THEN
2.       Pop all labels
3.       Forward the exposed frame on IFACE-OUT
4.   ELSE
5.       Drop the packet
```

Upon receiving on IFACE-IN an Ethernet frame with a destination address different than the interface address, a node N does:

```
1.   Push labels in CACHE on top of the frame Ethernet header
2.   Lookup the top label and proceed accordingly
```

The receiving interface must be configured in promiscuous mode in order to accept those Ethernet frames.

6.1.1.2. Static proxy for inner type IPv4

Upon receiving an MPLS packet with top label L, where L is an MPLS IPv4 static proxy segment, a node N does:

```
1.  IF payload type is IPv4 THEN
2.      Pop all labels
3.      Forward the exposed packet on IFACE-OUT towards S-ADDR
4.  ELSE
5.      Drop the packet
```

Upon receiving a non link-local IPv4 packet on IFACE-IN, a node N does:

```
1.  Decrement TTL and update checksum
2.  Push labels in CACHE on top of the packet IPv4 header
3.  Lookup the top label and proceed accordingly
```

6.1.1.3. Static proxy for inner type IPv6

Upon receiving an MPLS packet with top label L, where L is an MPLS IPv6 static proxy segment, a node N does:

```
1.  IF payload type is IPv6 THEN
2.      Pop all labels
3.      Forward the exposed packet on IFACE-OUT towards S-ADDR
4.  ELSE
5.      Drop the packet
```

Upon receiving a non link-local IPv6 packet on IFACE-IN, a node N does:

```
1.  Decrement Hop Limit
2.  Push labels in CACHE on top of the packet IPv6 header
3.  Lookup the top label and proceed accordingly
```

6.1.2. SRv6 pseudocode

6.1.2.1. Static proxy for inner type Ethernet

Upon receiving an IPv6 packet destined for S, where S is an IPv6 static proxy segment for Ethernet traffic, a node N does:

```
1.   IF ENH == 59 THEN                                        ;; Ref1
2.       Remove the (outer) IPv6 header and its extension headers
3.       Forward the exposed frame on IFACE-OUT
4.   ELSE
5.       Drop the packet
```

Ref1: 59 refers to "no next header" as defined by IANA allocation for Internet Protocol Numbers.

Upon receiving on IFACE-IN an Ethernet frame with a destination address different than the interface address, a node N does:

```
1.   Retrieve CACHE entry matching IFACE-IN and traffic type
2.   Push SRH with CACHE.LIST on top of the Ethernet header   ;; Ref2
3.   Push IPv6 header with
       SA = CACHE.SA
       DA = CACHE.LIST[0]                                     ;; Ref3
       Next Header = 43                                       ;; Ref4
4.   Set outer payload length and flow label
5.   Lookup outer DA in appropriate table and proceed accordingly
```

Ref2: Unless otherwise specified, the segments in CACHE.LIST should be encoded in reversed order, Segment Left and Last Entry values should be set of the length of CACHE.LIST minus 1, and Next Header should be set to 59.

Ref3: CACHE.LIST[0] represents the first IPv6 SID in CACHE.LIST.

Ref4: If CACHE.LIST contains a single entry, the SRH can be omitted and the Next Header value must be set to 59.

The receiving interface must be configured in promiscuous mode in order to accept those Ethernet frames.

6.1.2.2. Static proxy for inner type IPv4

Upon receiving an IPv6 packet destined for S, where S is an IPv6 static proxy segment for IPv4 traffic, a node N does:

```
1.   IF ENH == 4 THEN                                         ;; Ref1
2.       Remove the (outer) IPv6 header and its extension headers
3.       Forward the exposed packet on IFACE-OUT towards S-ADDR
4.   ELSE
5.       Drop the packet
```

Ref1: 4 refers to IPv4 encapsulation as defined by IANA allocation for Internet Protocol Numbers.

Upon receiving a non link-local IPv4 packet on IFACE-IN, a node N does:

```
1.  Decrement TTL and update checksum
2.  IF CACHE.SRH THEN                                          ;; Ref2
3.      Push CACHE.SRH on top of the existing IPv4 header
4.      Set NH value of the pushed SRH to 4
5.  Push outer IPv6 header with SA, DA and traffic class from CACHE
6.  Set outer payload length and flow label
7.  Set NH value to 43 if an SRH was added, or 4 otherwise
8.  Lookup outer DA in appropriate table and proceed accordingly
```

Ref2: CACHE.SRH represents the SRH defined in CACHE, if any, for the static SR proxy segment associated with IFACE-IN.

6.1.2.3. Static proxy for inner type IPv6

Upon receiving an IPv6 packet destined for S, where S is an IPv6 static proxy segment for IPv6 traffic, a node N does:

```
1.  IF ENH == 41 THEN                                          ;; Ref1
2.      Remove the (outer) IPv6 header and its extension headers
3.      Forward the exposed packet on IFACE-OUT towards S-ADDR
4.  ELSE
5.      Drop the packet
```

Ref1: 41 refers to IPv6 encapsulation as defined by IANA allocation for Internet Protocol Numbers.

Upon receiving a non-link-local IPv6 packet on IFACE-IN, a node N does:

```
1.  Decrement Hop Limit
2.  IF CACHE.SRH THEN                                          ;; Ref2
3.      Push CACHE.SRH on top of the existing IPv6 header
4.      Set NH value of the pushed SRH to 41
5.  Push outer IPv6 header with SA, DA and traffic class from CACHE
6.  Set outer payload length and flow label
7.  Set NH value to 43 if an SRH was added, or 41 otherwise
8.  Lookup outer DA in appropriate table and proceed accordingly
```

Ref2: CACHE.SRH represents the SRH defined in CACHE, if any, for the static SR proxy segment associated with IFACE-IN.

6.2. Dynamic SR proxy

The dynamic proxy is an improvement over the static proxy that dynamically learns the SR information before removing it from the incoming traffic. The same information can then be re-attached to the traffic returning from the SF. As opposed to the static SR proxy, no CACHE information needs to be configured. Instead, the dynamic SR proxy relies on a local caching mechanism on the node instantiating this segment. Therefore, a dynamic proxy segment cannot be the last segment in an SR SC policy. As mentioned at the beginning of Section 6, a different SR behavior should be used if the SF is meant to be the final destination of an SR SC policy.

Upon receiving a packet whose active segment matches a dynamic SR proxy function, the proxy node pops the top MPLS label or applies the SRv6 End behavior, then compares the updated SR information with the cache entry for the current segment. If the cache is empty or different, it is updated with the new SR information. The SR information is then removed and the inner packet is sent towards the SF.

The cache entry is not mapped to any particular packet, but instead to an SR SC policy identified by the receiving interface (IFACE-IN). Any non-link-local IP packet or non-local Ethernet frame received on that interface will be re-encapsulated with the cached headers as described in Section 6.1. The SF may thus drop, modify or generate new packets without affecting the proxy.

6.2.1. SR-MPLS pseudocode

The static proxy SR-MPLS pseudocode is augmented by inserting the following instructions between lines 1 and 2.

```
1.   IF top label S bit is 0 THEN
2.       Pop top label
3.       IF C(IFACE-IN) different from remaining labels THEN  ;; Ref1
4.           Copy all remaining labels into C(IFACE-IN)       ;; Ref2
5.   ELSE
6.       Drop the packet
```

Ref1: A TTL margin can be configured for the top label stack entry to prevent constant cache updates when multiple equal-cost paths with different hop counts are used towards the SR proxy node. In that case, a TTL difference smaller than the configured margin should not trigger a cache update (provided that the labels are the same).

Ref2: C(IFACE-IN) represents the cache entry associated to the dynamic SR proxy segment. It is identified with IFACE-IN in order to

efficiently retrieve the right SR information when a packet arrives on this interface.

In addition, the inbound policy should check that C(IFACE-IN) has been defined before attempting to restore the MPLS label stack, and drop the packet otherwise.

6.2.2. SRv6 pseudocode

The static proxy SRv6 pseudocode is augmented by inserting the following instructions between lines 1 and 2.

```
1.   IF NH=SRH & SL > 0 THEN
2.       Decrement SL and update the IPv6 DA with SRH[SL]
3.       IF C(IFACE-IN) different from IPv6 encaps THEN       ;; Ref1
4.           Copy the IPv6 encaps into C(IFACE-IN)            ;; Ref2
5.   ELSE
6.       Drop the packet
```

Ref1: "IPv6 encaps" represents the IPv6 header and any attached extension header.

Ref2: C(IFACE-IN) represents the cache entry associated to the dynamic SR proxy segment. It is identified with IFACE-IN in order to efficiently retrieve the right SR information when a packet arrives on this interface.

In addition, the inbound policy should check that C(IFACE-IN) has been defined before attempting to restore the IPv6 encapsulation, and drop the packet otherwise.

6.3. Shared memory SR proxy

The shared memory proxy is an SR endpoint behavior for processing SR-MPLS or SRv6 encapsulated traffic on behalf of an SR-unaware SF. This proxy behavior leverages a shared-memory interface with the SF in order to hide the SR information from an SR-unaware SF while keeping it attached to the packet. We assume in this case that the proxy and the SF are running on the same compute node. A typical scenario is an SR-capable vrouter running on a container host and forwarding traffic to virtual SFs isolated within their respective container.

More details will be added in a future revision of this document.

6.4. Masquerading SR proxy

The masquerading proxy is an SR endpoint behavior for processing SRv6 traffic on behalf of an SR-unaware SF. This proxy thus receives SR traffic that is formed of an IPv6 header and an SRH on top of an inner payload. The masquerading behavior is independent from the inner payload type. Hence, the inner payload can be of any type but it is usually expected to be a transport layer packet, such as TCP or UDP.

A masquerading SR proxy segment is associated with the following mandatory parameters:

- S-ADDR: Ethernet or IPv6 address of the SF
- IFACE-OUT: Local interface for sending traffic towards the SF
- IFACE-IN: Local interface receiving the traffic coming back from the SF

A masquerading SR proxy segment is thus defined for a specific SF and bound to a pair of directed interfaces or sub-interfaces on the proxy. As opposed to the static and dynamic SR proxies, a masquerading segment can be present at the same time in any number of SR SC policies and the same interfaces can be bound to multiple masquerading proxy segments. The only restriction is that a masquerading proxy segment cannot be the last segment in an SR SC policy.

The first part of the masquerading behavior is triggered when the proxy node receives an IPv6 packet whose Destination Address matches a masquerading proxy segment. The proxy inspects the IPv6 extension headers and substitutes the Destination Address with the last segment in the SRH attached to the IPv6 header, which represents the final destination of the IPv6 packet. The packet is then sent out towards the SF.

The SF receives an IPv6 packet whose source and destination addresses are respectively the original source and final destination. It does not attempt to inspect the SRH, as RFC8200 specifies that routing extension headers are not examined or processed by transit nodes. Instead, the SF simply forwards the packet based on its current Destination Address. In this scenario, we assume that the SF can only inspect, drop or perform limited changes to the packets. For example, Intrusion Detection Systems, Deep Packet Inspectors and non-NAT Firewalls are among the SFs that can be supported by a masquerading SR proxy. Variants of the masquerading behavior are

defined in Section 6.4.2 and Section 6.4.3 to support a wider range of SFs.

The second part of the masquerading behavior, also called de-masquerading, is an inbound policy attached to the proxy interface receiving the traffic returning from the SF, IFACE-IN. This policy inspects the incoming traffic and triggers a regular SRv6 endpoint processing (End) on any IPv6 packet that contains an SRH. This processing occurs before any lookup on the packet Destination Address is performed and it is sufficient to restore the right active segment as the Destination Address of the IPv6 packet.

6.4.1. SRv6 masquerading proxy pseudocode

Masquerading: Upon receiving a packet destined for S, where S is an IPv6 masquerading proxy segment, a node N processes it as follows.

```
1.   IF NH=SRH & SL > 0 THEN
2.       Update the IPv6 DA with SRH[0]
3.       Forward the packet on IFACE-OUT
4.   ELSE
5.       Drop the packet
```

De-masquerading: Upon receiving a non-link-local IPv6 packet on IFACE-IN, a node N processes it as follows.

```
1.   IF NH=SRH & SL > 0 THEN
2.       Decrement SL
3.       Update the IPv6 DA with SRH[SL]                    ;; Ref1
4.       Lookup DA in appropriate table and proceed accordingly
```

Ref2: This pseudocode can be augmented to support the Penultimate Segment Popping (PSP) endpoint flavor. The exact pseudocode modification are provided in [I-D.filsfils-spring-srv6-network-programming].

6.4.2. Variant 1: Destination NAT

SFs modifying the destination address in the packets they process, such as NATs, can be supported by a masquerading proxy with the following modification to the de-masquerading pseudocode.

De-masquerading - NAT: Upon receiving a non-link-local IPv6 packet on IFACE-IN, a node N processes it as follows.

```
1.  IF NH=SRH & SL > 0 THEN
2.      Update SRH[0] with the IPv6 DA
3.      Decrement SL
4.      Update the IPv6 DA with SRH[SL]
5.      Lookup DA in appropriate table and proceed accordingly
```

6.4.3. Variant 2: Caching

SFs generating packets or acting as endpoints for transport connections can be supported by adding a dynamic caching mechanism similar to the one described in Section 6.2.

More details will be added in a future revision of this document.

7. Metadata

7.1. MPLS data plane

Since the MPLS encapsulation has no explicit protocol identifier field to indicate the protocol type of the MPLS payload, how to indicate the presence of metadata (i.e., the NSH which is only used as a metadata containner) in an MPLS packet is a potential issue to be addressed. One possible way to address the above issue is: SFFs allocate two different labels for a given SF, one indicates the presence of NSH while the other indicates the absence of NSH. This approach has no change to the current MPLS architecture but it would require more than one label binding for a given SF. Another possible way is to introduce a protocol identifier field within the MPLS packet as described in [I-D.xu-mpls-payload-protocol-identifier].

More details about how to contain metadata within an MPLS packet would be considered in the future version of this draft.

7.2. IPv6 data plane

7.2.1. SRH TLV objects

The IPv6 SRH TLV objects are designed to carry all sorts of metadata. In particular, [I-D.ietf-6man-segment-routing-header] defines the NSH carrier TLV as a container for NSH metadata.

TLV objects can be imposed by the ingress edge router that steers the traffic into the SR SC policy.

An SR-aware SF may impose, modify or remove any TLV object attached to the first SRH, either by directly modifying the packet headers or via a control channel between the SF and its forwarding plane.

An SR-aware SF that re-classifies the traffic and steers it into a new SR SC policy (e.g. DPI) may attach any TLV object to the new SRH.

Metadata imposition and handling will be further discussed in a future version of this document.

7.2.2. SRH tag

The SRH tag identifies a packet as part of a group or class of packets [I-D.ietf-6man-segment-routing-header].

In an SFC context, this field can be used as a simple man's metadata to encode additional information in the SRH.

8. Implementation status

The static SR proxy is available for SR-MPLS and SRv6 on various Cisco hardware and software platforms. Furthermore, the following proxies are available on open-source software.

		VPP	Linux
MPLS	Static proxy	Available	In progress
	Dynamic proxy	In progress	In progress
	Shared memory proxy	In progress	In progress
SRv6	Static proxy	Available	In progress
	Dynamic proxy - Inner type Ethernet	In progress	In progress
	Dynamic proxy - Inner type IPv4	Available	Available
	Dynamic proxy - Inner type IPv6	Available	Available
	Shared memory proxy	In progress	In progress
	Masquerading proxy	Available	Available
	Masquerading proxy - NAT variant	In progress	In progress
	Masquerading proxy - Cache variant	In progress	In progress

Open-source implementation status table

9. Related works

The Segment Routing solution addresses a wide problem that covers both topological and service chaining policies. The topological and service instructions can be either deployed in isolation or in combination. SR has thus a wider applicability than the architecture defined in [RFC7665]. Furthermore, the inherent property of SR is a stateless network fabric. In SR, there is no state within the fabric to recognize a flow and associate it with a policy. State is only present at the ingress edge of the SR domain, where the policy is encoded into the packets. This is completely different from other proposals such as [I-D.ietf-sfc-nsh] and the MPLS label swapping mechanism described in [I-D.farrel-mpls-sfc], which rely on state configured at every hop of the service chain.

10. IANA Considerations

This document has no actions for IANA.

11. Security Considerations

The security requirements and mechanisms described in [I-D.ietf-spring-segment-routing] and [I-D.ietf-6man-segment-routing-header] also apply to this document.

Furthermore, it is fundamental to the SFC design that the classifier is a trusted resource which determines the processing that the packet will be subject to, including for example the firewall. Where an SF is not SR-aware the packet may exist as an IP packet, however this is an intrinsic part of the SFC design which needs to define how a packet is protected in that environment. Where a tunnel is used to link two non-MPLS domains, the tunnel design needs to specify how it is secured.

Thus the security vulnerabilities are addressed in the underlying technologies used by this design, which itself does not introduce any new security vulnerabilities.

12. Acknowledgements

The authors would like to thank Loa Andersson, Andrew G. Malis, Adrian Farrel, Alexander Vainshtein and Joel M. Halpern for their valuable comments and suggestions on the document.

13. Contributors

P. Camarillo (Cisco), B. Peirens (Proximus), D. Steinberg (Steinberg Consulting), A. AbdelSalam (Gran Sasso Science Institute), G. Dawra (Cisco), S. Bryant (Huawei), H. Assarpour (Broadcom), H. Shah (Ciena), L. Contreras (Telefonica I+D), J. Tantsura (Individual), M. Vigoureux (Nokia) and J. Bhattacharya (Cisco) 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, <https://www.rfc-editor.org/info/rfc2119>.

14.2. Informative References

[I-D.dawra-idr-bgp-sr-service-chaining]
Dawra, G., Filsfils, C., daniel.bernier@bell.ca, d., Uttaro, J., Decraene, B., Elmalky, H., Xu, X., Clad, F., and K. Talaulikar, "BGP Control Plane Extensions for Segment Routing based Service Chaining", draft-dawra-idr-bgp-sr-service-chaining-02 (work in progress), January 2018.

[I-D.farrel-mpls-sfc]
Farrel, A., Bryant, S., and J. Drake, "An MPLS-Based Forwarding Plane for Service Function Chaining", draft-farrel-mpls-sfc-02 (work in progress), October 2017.

[I-D.filsfils-spring-segment-routing-policy]
Filsfils, C., Sivabalan, S., Raza, K., Liste, J., Clad, F., Talaulikar, K., Hegde, S., daniel.voyer@bell.ca, d., Lin, S., bogdanov@google.com, b., Horneffer, M., Steinberg, D., Decraene, B., Litkowski, S., and P. Mattes, "Segment Routing Policy for Traffic Engineering", draft-filsfils-spring-segment-routing-policy-04 (work in progress), December 2017.

[I-D.filsfils-spring-srv6-network-programming]
Filsfils, C., Leddy, J., daniel.voyer@bell.ca, d., daniel.bernier@bell.ca, d., Steinberg, D., Raszuk, R., Matsushima, S., Lebrun, D., Decraene, B., Peirens, B., Salsano, S., Naik, G., Elmalky, H., Jonnalagadda, P., Sharif, M., Ayyangar, A., Mynam, S., Henderickx, W., Bashandy, A., Raza, K., Dukes, D., Clad, F., and P. Camarillo, "SRv6 Network Programming", draft-filsfils-spring-srv6-network-programming-03 (work in progress), December 2017.

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

[I-D.ietf-sfc-nsh]
Quinn, P., Elzur, U., and C. Pignataro, "Network Service Header (NSH)", draft-ietf-sfc-nsh-28 (work in progress), November 2017.

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-14 (work in progress), December 2017.

[I-D.ietf-spring-segment-routing-mpls]
Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with MPLS data plane", draft-ietf-spring-segment-routing-mpls-11 (work in progress), October 2017.

[I-D.xu-mpls-payload-protocol-identifier]
Xu, X., Assarpour, H., and S. Ma, "MPLS Payload Protocol Identifier", draft-xu-mpls-payload-protocol-identifier-04 (work in progress), January 2018.

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

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

[RFC4817] Townsley, M., Pignataro, C., Wainner, S., Seely, T., and J. Young, "Encapsulation of MPLS over Layer 2 Tunneling Protocol Version 3", RFC 4817, DOI 10.17487/RFC4817, March 2007, <https://www.rfc-editor.org/info/rfc4817>.

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

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

Authors' Addresses

Francois Clad (editor)
Cisco Systems, Inc.
France

Email: fclad@cisco.com

Xiaohu Xu (editor)
Huawei

Email: xuxiaohu@huawei.com

Clarence Filsfils
Cisco Systems, Inc.
Belgium

Email: cf@cisco.com

Daniel Bernier
Bell Canada
Canada

Email: daniel.bernier@bell.ca

Bruno Decraene
Orange
France

Email: bruno.decraene@orange.com

Shaowen Ma
Juniper

Email: mashaowen@gmail.com

Chaitanya Yadlapalli
AT&T
USA

Email: cy098d@att.com

Wim Henderickx
Nokia
Belgium

Email: wim.henderickx@nokia.com

Stefano Salsano
Universita di Roma "Tor Vergata"
Italy

Email: stefano.salsano@uniroma2.it