SPRING
Internet-Draft
Intended status: Standards Track
Expires: 27 August 2025

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

23 February 2025

# Service Programming with Segment Routing
# draft-ietf-spring-sr-service-programming-11

## Abstract

This document defines data plane functionality required to implement service segments and achieve service programming in SR-enabled MPLS and IPv6 networks, as described in the Segment Routing architecture.

## 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 27 August 2025.

Table of Contents

1. Introduction . . . . . 3
2. Terminology . . . . . 3
3. Classification and Steering . . . . . 4
4. Service Segments . . . . . 4
4.1. SR-Aware Services . . . . . 5
4.2. SR-Unaware Services . . . . . 6
5. SR Service Policies . . . . . 6
5.1. SR-MPLS Data Plane . . . . . 8
5.2. SRv6 Data Plane . . . . . 9
6. SR Proxy Behaviors . . . . . 10
6.1. Static SR Proxy . . . . . 13
6.1.1. SR-MPLS Pseudocode . . . . . 15
6.1.2. SRv6 Pseudocode . . . . . 16
6.2. Dynamic SR Proxy . . . . . 22
6.2.1. SR-MPLS Pseudocode . . . . . 23
6.2.2. SRv6 Pseudocode . . . . . 23
6.3. Shared Memory SR Proxy . . . . . 24
6.4. Masquerading SR Proxy . . . . . 24
6.4.1. SRv6 Masquerading Proxy Pseudocode . . . . . 26
6.4.2. Destination NAT Flavor . . . . . 27
6.4.3. Caching Flavor . . . . . 27
7. Metadata . . . . . 28
7.1. MPLS Data Plane . . . . . 28
7.2. IPv6 Data Plane . . . . . 29
7.2.1. SRH TLV Objects . . . . . 29
7.2.2. SRH Tag . . . . . 30
8. Implementation Status . . . . . 30
8.1. SR-Aware Services . . . . . 30
8.2. Proxy Behaviors . . . . . 31
9. Related Works . . . . . 31
10. IANA Considerations . . . . . 32
10.1. SRv6 Endpoint Behaviors . . . . . 32

10.2. Segment Routing Header TLVs . . . . . 32
11. Security Considerations . . . . . 32
12. Acknowledgements . . . . . 32
13. References . . . . . 32
13.1. Normative References . . . . . 33
13.2. Informative References . . . . . 33
Contributors . . . . . 34
Authors' Addresses . . . . . 36

## 1. Introduction

Segment Routing (SR) [RFC8402] 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, but also steer them through Virtual Network Functions (VNFs) or physical service appliances available in the network.

In an SR network, each of these services, running either on a physical appliance or in a virtual environment, are associated with a segment identifier (SID). These service SIDs are then leveraged as part of a SID-list to steer packets through the corresponding services. Service SIDs may be combined together in a SID-list to achieve service programming, but also with other types of segments as defined in [RFC8402]. SR thus provides a fully integrated solution for overlay, underlay and service programming. Furthermore, the IPv6 instantiation of SR (SRv6) [RFC8986] supports metadata transportation in the Segment Routing Header [RFC8754], either natively in the tag field or with extensions such as TLVs.

This document describes how a service can be associated with a SID, including legacy services with no SR capabilities, and how these service SIDs are integrated within an SR policy. The definition of an SR Policy and the traffic steering mechanisms are covered in [RFC9256] and hence outside the scope of this document.

The definition of control plane components, such as service segment discovery, is outside the scope of this data plane document. For reference, the option of using BGP extensions to support SR service programming is proposed in [I-D.dawra-idr-bgp-sr-service-chaining].

## 2. Terminology

This document leverages the terminology proposed in [RFC8402], [RFC8660], [RFC8754], [RFC8986] and [RFC9256]. It also introduces the following new terms.

Service segment: A segment associated with a service. The service may either run on a physical appliance or in a virtual environment such as a virtual machine or container.

SR-aware service: A service that is fully capable of processing SR traffic. An SR-aware service can be directly associated with a service segment.

SR-unaware service: A service that is unable to process SR traffic or may behave incorrectly due to presence of SR information in the packet headers. An SR-unaware service can be associated with a service segment through an SR proxy function.

## 3. Classification and Steering

Classification and steering mechanisms are defined in section 8 of [RFC9256] and are independent from the purpose of the SR policy. From the perspective of a headend node classifying and steering traffic into an SR policy, there is no difference whether this policy contains IGP, BGP, peering, VPN or service segments, or any combination of these.

As documented in the above reference, traffic is classified when entering an SR domain. The SR policy headend 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 [RFC8754] can be used to identify and classify packets sharing the same set of properties. Classified traffic is then steered into the appropriate SR policy and forwarded as per the SID-list(s) of the active candidate path.

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 SRH.

## 4. Service Segments

In the context of this document, the term service refers to a physical appliance running on dedicated hardware, a virtualized service inside an isolated environment such as a Virtual Machine (VM), container or namespace, or any process running on a compute element. A service 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 a service.

The execution of a service can be integrated as part of an SR policy by assigning a segment identifier, or SID, to the service and including this service SID in the SR policy SID-list. Such a service SID may be of local or global significance. In the former case, other segments, such as prefix or adjacency segments, can be used to steer the traffic up to the node where the service segment is instantiated. In the latter case, the service is directly reachable from anywhere in the routing domain. This is realized with SR-MPLS by assigning a SID from the global label block ([RFC8660]), or with SRv6 by advertising the SID locator in the routing protocol ([RFC8986]). 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 services 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 services.

### 4.1. SR-Aware Services

An SR-aware service 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 service's 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.

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

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

If the service 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 service-specific functions is out of the scope of this document.

SR-aware services also enable advanced network programming functionalities such as conditional branching and jumping to arbitrary SIDs in the segment list. In addition, SRv6 provides several ways of passing and exchanging information between services (e.g., SID arguments, tag field and TLVs). An example scenario involving these features is described in [IFIP18], which discusses the implementation of an SR-aware Intrusion Detection System.

Examples of SR-aware services are provided in section Section 8.1.

### 4.2. SR-Unaware Services

Any service that does not meet the above criteria for SR-awareness is considered as SR-unaware.

An SR-unaware service 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 services in an SR policy, it is thus required to remove the SR information as well as any other encapsulation header before the service receives the packet, or to alter it in such a way that the service can correctly process the packet.

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

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

## 5. SR Service Policies

An SR service policy is an SR policy, as defined in [RFC9256], that includes at least one service. This service is represented in the SID-list by its associated service SID. In case the policy should include several services, the service traversal order is indicated by the relative position of each service SID in the SID-list. Using the mechanisms described in [RFC9256], it is possible to load balance the traffic over several services, or instances of the same service, by associating with the SR service policy a weighted set of SID-lists, each containing a possible sequence of service SIDs to be traversed. Similarly, several candidate paths can be specified for the SR service policy, each with its own set of SID-lists, for resiliency

purposes.

Furthermore, binding SIDs (BSIDs) [RFC8402] can be leveraged in the context of service policies to reduce the number of SIDs imposed by the headend, provide opacity between domains and improve scalability. For example, a network operator may want a policy in its core domain to include services that are running in one of its datacenters. One option is to define an SR policy at ingress edge of the core domain that explicitly includes all the SIDs needed to steer the traffic through the core and in the DC, but that may result in a long SID-list and requires to update the ingress edge configuration every time the DC part of the policy is modified. Alternatively, a separate policy can be defined at the ingress edge of the datacenter with only the SIDs that needs to be executed there and its BSID included in the core domain policy. That BSID remains stable when the DC policy is modified and can even be shared among several core domain policies that would require the same type of processing in the DC.

This section describes how services can be integrated within an SR-MPLS or SRv6 service policy.

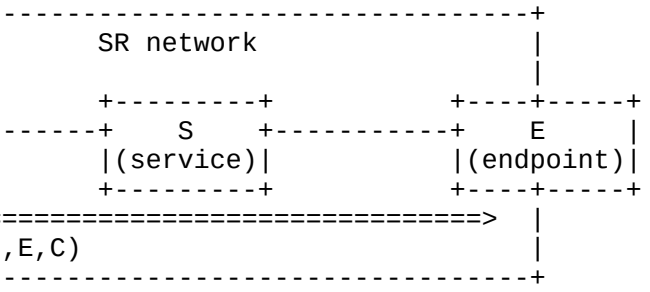



Figure 1: SR service policy

Figure 1 illustrates a basic SR service policy instantiated on a headend node H towards an endpoint E and traversing a service S. The SR policy may also include additional requirements, such as traffic engineering or VPN. On the head-end H, the SR policy P1 is created with a color C and endpoint E and associated with an SR path that can either be explicitly configured, dynamically computed on H or provisioned by a network controller.

In its most basic form, the SR policy P1 would be resolved into the SID-list < SID(S), SID(E) >. This is assuming that SID(S) and SID(E) are directly reachable from H and S, respectively, and that the forwarding path meets the policy requirement. However, depending on the dataplane and the segments available in the network, additional SIDs may be required to enforce the SR policy.

This model applies regardless of the SR-awareness of the service. If it is SR-unaware, then S simply represents the proxy that takes care of transmitting the packet to the actual service.

Traffic can then be steered into this policy using any of the mechanisms described in [RFC9256].

The following subsections describe the specificities of each SR dataplane.

## 5.1. SR-MPLS Data Plane

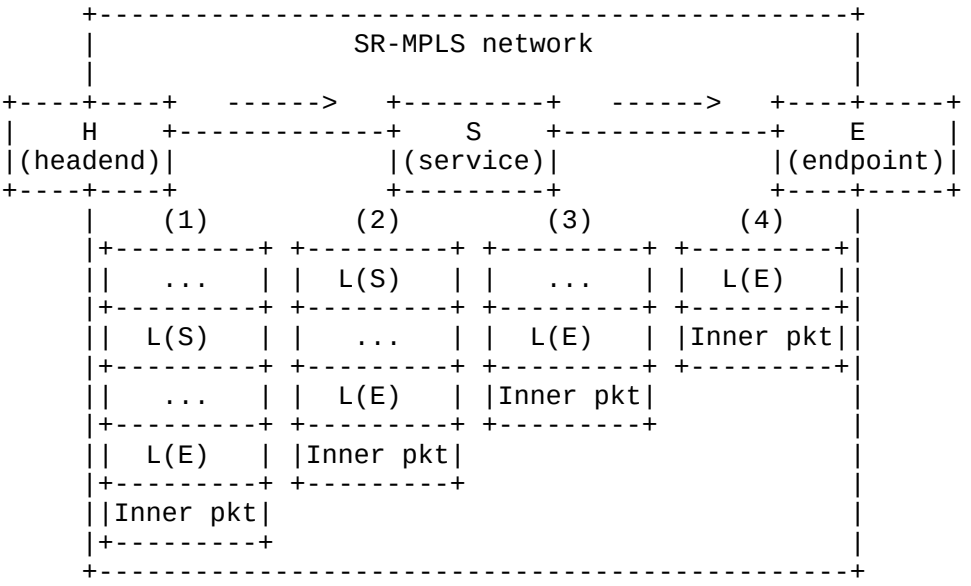



Figure 2: Packet walk in an SR-MPLS network

In an SR-MPLS network, the SR policy SID-list is encoded as a stack of MPLS labels [RFC8660] and pushed on top of the packet.

In the example shown on Figure 2, the SR policy should steer the traffic from the head-end H to the endpoint E via a service S. This translates into an MPLS label stack that includes at least a label L(S) associated to service S and a label L(E) associated to the endpoint E. The label stack may also include additional intermediate SIDs if these are required for traffic engineering (e.g., to encode a low latency path between H and S and / or between S and E) or simply for reachability purposes. Indeed, the service SID L(S) may be taken from the global or local SID block of node S and, in the latter case, one or more SIDs might be needed before L(S) in order for the packet

to reach node S (e.g., a prefix-SID of S), where L(S) can be interpreted. The same applies for the SID L(E) at the SR policy endpoint.

Special consideration must be taken into account when using Local SIDs for service identification due to increased label stack depth and the associated impacts.

When the packet arrives at S, this node determines the MPLS payload type and the appropriate behavior for processing the packet based on the semantic locally associated to the top label L(S). If S is an SR-aware service, the SID L(S) may provide additional context or indication on how to process the packet (e.g., a firewall SID may indicate which rule set should be applied onto the packet). If S is a proxy in front of an SR-unaware service, L(S) indicates how and to which service attached to this proxy the packet should be transmitted. At some point in the process, L(S) is also popped from the label stack in order to expose the next SID, which may be L(E) or another intermediate SID.

## 5.2. SRv6 Data Plane

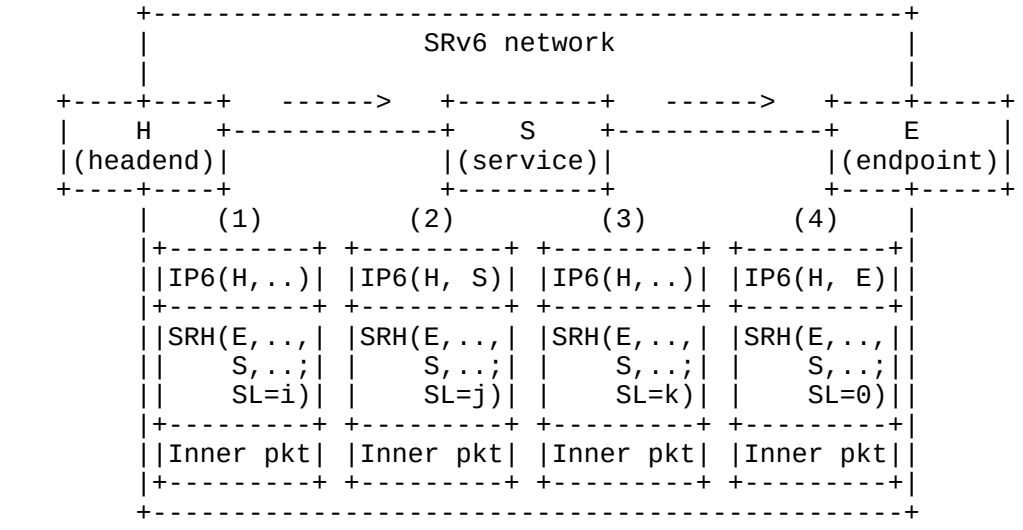



Figure 3: Packet walk in an SRv6 network

In an SRv6 network, the SR Policy is encoded into the packet as an IPv6 header possibly followed by a Segment Routing Header (SRH) [RFC8754].

In the example shown on Figure 3, the SR policy should steer the traffic from the head-end H to the endpoint E via a service S. This translates into Segment-List that includes at least a segment SID(S) to the service, or service proxy, S and a segment SID(E) to the endpoint E. The Segment-List may also include additional intermediate SIDs if these are required for traffic engineering (e.g., the encode a low latency path between H and S and / or between S and E) or simply for reachability purposes. Indeed, the service SID locator may or may not be advertised in the routing protocol and, in the latter case, one or more SIDs might be needed before SID(S) in order to bring the packet up to node S, where SID(S) can be interpreted. The same applies for the segment SID(E) at the SR policy endpoint.

When the packet arrives at S, this node determines how to process the packet based on the semantic locally associated to the active segment SID(S). If S is an SR-aware service, then SID(S) may provide additional context or indication on how to process the packet (e.g., a firewall SID may indicate which rule set should be applied onto the packet). If S is a proxy in front of an SR-unaware service, SID(S) indicates how and to which service attached to this proxy the packet should be transmitted. At some point in the process, the SRv6 End function is also applied in order to make the next SID, which may be SID(E) or another intermediate SID, active.

The "Inner pkt" on Figure 3 represents the SRv6 payload, which may be an encapsulated IP packet, an Ethernet frame or a transport-layer payload, for example.

## 6. SR Proxy Behaviors

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

A service 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 services at intermediate hops in the segment list. In case an SR-unaware service 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 service (MPLS PHP, SRv6 decapsulation behavior or PSP flavor).

As illustrated on Figure 4, the generic behavior of an SR proxy has two parts. The first part is in charge of passing traffic from the network to the service. It intercepts the SR traffic destined for the service via a locally instantiated service segment, modifies it in such a way that it appears as non-SR traffic to the service, then sends it out on a given interface, IFACE-OUT, connected to the service. The second part receives the traffic coming back from the service on IFACE-IN, restores the SR information and forwards it according to the next segment in the list. IFACE-OUT and IFACE-IN are respectively the proxy interface used for sending traffic to the service and the proxy interface that receives the traffic coming back from the service. These can be physical interfaces or sub-interfaces (VLANs) and, unless otherwise stated, IFACE-OUT and IFACE-IN can represent the same interface.

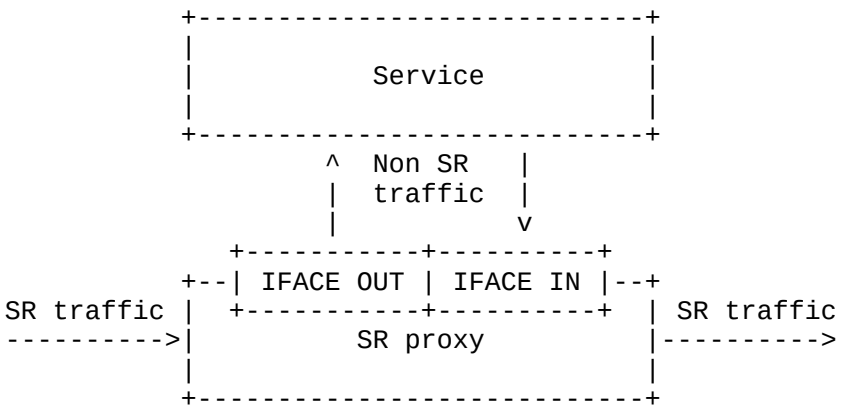



Figure 4: Generic SR proxy

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 service behavior and the traffic type. It is also possible to use different proxy mechanisms within the same service policy.

| | | Static | Dynamic | Shared mem. | Masquerading |
|---|---|---|---|---|---|
| SR flavors | SR-MPLS | Y | Y | Y | - |
| | Inline SRv6 | P | P | P | Y |
| | SRv6 encapsulation | Y | Y | Y | - |
| Chain agnostic configuration | | N | N | Y | Y |
| Transparent to chain changes | | N | Y | Y | Y |
| Service 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 |
| | Packet re-ordering | Y | Y | Y | Y |
| | Transport endpoint | Y | Y | cache | cache |
| Supported traffic | Ethernet | Y | Y | Y | - |
| | IPv4 | Y | Y | Y | - |
| | IPv6 | Y | Y | Y | Y |

Figure 5: SR proxy summary

Note: The use of a shared memory proxy requires both the service (VNF) 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 service. 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
* NH-ADDR: Next hop Ethernet address (only for inner type IPv4 and IPv6)
* IFACE-OUT: Local interface for sending traffic towards the service
* IFACE-IN: Local interface receiving the traffic coming back from the service
* CACHE: SR information to be attached on the traffic coming back from the service, 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 service, 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 service is to be used as part of a bi-directional SR service policy. If the proxy and the service 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 service. 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 service, 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 of 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. An MPLS TTL or IPv6 Hop Limit value may also be configured in CACHE. If it is not, the proxy should set these values according to the node’s default setting for MPLS or IPv6 encapsulation.

In this scenario, there are no restrictions on the operations that can be performed by the service 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 service into an SRv6 policy. However, a static SR proxy segment can be used in only one service policy 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 service 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 service 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 service policy, a different SR proxy segment and receiving interface are required for the return direction.

The static proxy behavior may also be used for sending traffic through "bump in the wire" services that are transparent to the IP and Ethernet layers. This type of processing is assumed when the inner traffic type is Ethernet, since the original destination address of the Ethernet frame is preserved when the packet is steered into the SR Policy and likely associated with a node downstream of the policy tail-end. In case the inner type is IP (IPv4 or IPv6), the NH-ADDR parameter may be set to a dummy or broadcast Ethernet address, or simply to the address of the proxy receiving interface (IFACE-IN).

### 6.1.1. SR-MPLS Pseudocode

#### 6.1.1.1. Static Proxy for Inner Type Ethernet

When processing an MPLS packet whose top label matches a locally instantiated MPLS static proxy SID for Ethernet traffic, the following pseudocode is executed.

```
S01. POP all labels in the MPLS label stack.
S02. Submit the frame to the Ethernet module for transmission via
     interface IFACE-OUT.
```

Figure 6: SID processing for MPLS static proxy (Ethernet)

When processing an Ethernet frame received on the interface IFACE-IN and with a destination MAC address that is neither a broadcast address nor matches the address of IFACE-IN, the following pseudocode is executed.

```
S01. Retrieve the CACHE entry associated with IFACE-IN.
S02. If the CACHE entry is not empty {
S03.   Remove the preamble or Frame Check Sequence (FCS).
S04.   PUSH all labels from the retrieved CACHE entry.
S05.   Submit the packet to the MPLS module for transmission as per
       the top label in the MPLS label stack.
S06. }
```

Figure 7: Inbound policy for MPLS static proxy (Ethernet)

#### 6.1.1.2. Static Proxy for Inner Type IPv4

When processing an MPLS packet whose top label matches a locally instantiated MPLS static proxy SID for IPv4 traffic, the following pseudocode is executed.

```
S01. POP all labels in the MPLS label stack.
S02. Submit the packet to the IPv4 module for transmission on
     interface IFACE-OUT via NH-ADDR.
```

Figure 8: SID processing for MPLS static proxy (IPv4)

When processing an IPv4 packet received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN, the following pseudocode is executed.

```
S01. Retrieve the CACHE entry associated with IFACE-IN.
S02. If the CACHE entry is not empty {
S03.   Decrement the TTL and adjust the checksum accordingly.
S04.   PUSH all labels from the retrieved CACHE entry.
S05.   Submit the packet to the MPLS module for transmission as per
       the top label in the MPLS label stack.
S06. }
```

Figure 9: Inbound policy for MPLS static proxy (IPv4)

#### 6.1.1.3. Static Proxy for Inner Type IPv6

When processing an MPLS packet whose top label matches a locally instantiated MPLS static proxy SID for IPv6 traffic, the following pseudocode is executed.

```
S01. POP all labels in the MPLS label stack.
S02. Submit the packet to the IPv6 module for transmission on
     interface IFACE-OUT via NH-ADDR.
```

Figure 10: SID processing for MPLS static proxy (IPv6)

When processing an IPv6 packet received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN, the following pseudocode is executed.

```
S01. Retrieve the CACHE entry associated with IFACE-IN.
S02. If the CACHE entry is not empty {
S03.   Decrement the Hop Limit.
S04.   PUSH all labels from the retrieved CACHE entry.
S05.   Submit the packet to the MPLS module for transmission as per
       the top label in the MPLS label stack.
S06. }
```

Figure 11: Inbound policy for MPLS static proxy (IPv6)

### 6.1.2. SRv6 Pseudocode

#### 6.1.2.1. Static Proxy for Inner Type Ethernet

When processing an IPv6 packet matching a FIB entry locally instantiated as an SRv6 static proxy SID for Ethernet traffic, the following pseudocode is executed.

```
S01. When an SRH is processed {
S02.   If (Segments Left == 0) {
S03.     Proceed to process the next header in the packet.
S04.   }
S05.   If (IPv6 Hop Limit <= 1) {
S06.     Send an ICMP Time Exceeded message to the Source Address,
         Code 0 (hop limit exceeded in transit),
         Interrupt packet processing and discard the packet.
S07.   }
S08.   max_last_entry = (Hdr Ext Len / 2) - 1
S09.   If ((Last Entry > max_last_entry) or
           (Segments Left > (Last Entry + 1))) {
S10.     Send an ICMP Parameter Problem message to the Source Address,
         Code 0 (Erroneous header field encountered),
         Pointer set to the Segments Left field,
         Interrupt packet processing and discard the packet.
S11.   }
S12.   Decrement Hop Limit by 1.
S13.   Decrement Segments Left by 1.
S14.   Copy Segment List[Segments Left] from the SRH to the
       Destination Address of the IPv6 header.
S15.   If (Upper-layer header type != 143 (Ethernet)) {
S16.     Resubmit the packet to the IPv6 module for transmission to
         the new destination.
S17.   }
S18.   Perform IPv6 decapsulation.
S19.   Submit the frame to the Ethernet module for transmission via
       interface IFACE-OUT.
S20. }
```

Figure 12: SID processing for SRv6 static proxy (Ethernet)

S15: 143 (Ethernet) refers to the value assigned by IANA for "Ethernet" in the "Internet Protocol Numbers" registry.

When processing the Upper-layer header of a packet matching a FIB entry locally instantiated as an SRv6 static proxy SID for Ethernet traffic, the following pseudocode is executed.

```
S01. If (Upper-layer header type != 143 (Ethernet)) {
S02.   Process as per [RFC8986] Section 4.1.1
S03. }
S04. Perform IPv6 decapsulation.
S05. Submit the frame to the Ethernet module for transmission via
     interface IFACE-OUT.
```

Figure 13: Upper-layer header processing for SRv6 static proxy (Ethernet)

When processing an Ethernet frame received on the interface IFACE-IN and with a destination MAC address that is neither a broadcast address nor matches the address of IFACE-IN, the following pseudocode is executed.

```
S01. Retrieve the CACHE entry associated with IFACE-IN.
S02. If the CACHE entry is not empty {
S03.   Remove the preamble or Frame Check Sequence (FCS).
S04.   Perform IPv6 encapsulation with an SRH
         Source Address of the IPv6 header is set to CACHE.SA,
         Destination Address of the IPv6 header is set to
         CACHE.LIST[0],
         Next Header of the SRH is set to 143 (Ethernet),
         Segment List of the SRH is set to CACHE.LIST.
S05.   Submit the packet to the IPv6 module for transmission to the
       next destination.
S06. }
```

Figure 14: Inbound policy for SRv6 static proxy (Ethernet)

S04: CACHE.LIST[0] represents the first entry in CACHE.LIST. Unless a local configuration indicates otherwise, the SIDs in CACHE.LIST should be encoded in the Segment List field in reversed order, the Segment Left and Last Entry values should be set of the length of CACHE.LIST minus 1. If CACHE.LIST contains a single entry, the SRH can be omitted and the Next Header field of the IPv6 header set to 143 (Ethernet).

#### 6.1.2.2. Static Proxy for Inner Type IPv4

When processing an IPv6 packet matching a FIB entry locally instantiated as an SRv6 static proxy SID for IPv4 traffic, the following pseudocode is executed.

```
S01. When an SRH is processed {
S02.   If (Segments Left == 0) {
S03.     Proceed to process the next header in the packet.
S04.   }
S05.   If (IPv6 Hop Limit <= 1) {
S06.     Send an ICMP Time Exceeded message to the Source Address,
         Code 0 (hop limit exceeded in transit),
         Interrupt packet processing and discard the packet.
S07.   }
S08.   max_last_entry = (Hdr Ext Len / 2) - 1
S09.   If ((Last Entry > max_last_entry) or
           (Segments Left > (Last Entry + 1))) {
S10.     Send an ICMP Parameter Problem message to the Source Address,
         Code 0 (Erroneous header field encountered),
         Pointer set to the Segments Left field,
         Interrupt packet processing and discard the packet.
S11.   }
S12.   Decrement Hop Limit by 1.
S13.   Decrement Segments Left by 1.
S14.   Copy Segment List[Segments Left] from the SRH to the
       Destination Address of the IPv6 header.
S15.   If (Upper-layer header type != 4 (IPv4)) {
S16.     Resubmit the packet to the IPv6 module for transmission to
         the new destination.
S17.   }
S18.   Perform IPv6 decapsulation.
S19.   Submit the packet to the IPv4 module for transmission on
       interface IFACE-OUT via NH-ADDR.
S20. }
```

Figure 15: SID processing for SRv6 static proxy (IPv4)

When processing the Upper-layer header of a packet matching a FIB entry locally instantiated as an SRv6 static proxy SID for IPv4 traffic, the following pseudocode is executed.

```
S01. If (Upper-layer header type != 4 (IPv4)) {
S02.   Process as per [RFC8986] Section 4.1.1
S03. }
S04. Perform IPv6 decapsulation.
S05. Submit the packet to the IPv4 module for transmission on
     interface IFACE-OUT via NH-ADDR.
```

Figure 16: Upper-layer header processing for SRv6 static proxy (IPv4)

When processing an IPv4 packet received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN, the following pseudocode is executed.

```
S01. Retrieve the CACHE entry associated with IFACE-IN.
S02. If the CACHE entry is not empty {
S03.   Decrement the TTL and adjust the checksum accordingly.
S04.   Perform IPv6 encapsulation with an SRH
         Source Address of the IPv6 header is set to CACHE.SA,
         Destination Address of the IPv6 header is set to
         CACHE.LIST[0],
         Next Header of the SRH is set to 4 (IPv4),
         Segment List of the SRH is set to CACHE.LIST.
S05.   Submit the packet to the IPv6 module for transmission to the
       next destination.
S06. }
```

Figure 17: Inbound policy for SRv6 static proxy (IPv4)

S04: CACHE.LIST[0] represents the first entry in CACHE.LIST. Unless a local configuration indicates otherwise, the SIDs in CACHE.LIST should be encoded in the Segment List field in reversed order, the Segment Left and Last Entry values should be set of the length of CACHE.LIST minus 1. If CACHE.LIST contains a single entry, the SRH can be omitted and the Next Header field of the IPv6 header set to 4 (IPv4).

#### 6.1.2.3. Static Proxy for Inner Type IPv6

When processing an IPv6 packet matching a FIB entry locally instantiated as an SRv6 static proxy SID for IPv6 traffic, the following pseudocode is executed.

```
S01. When an SRH is processed {
S02.   If (Segments Left == 0) {
S03.     Proceed to process the next header in the packet.
S04.   }
S05.   If (IPv6 Hop Limit <= 1) {
S06.     Send an ICMP Time Exceeded message to the Source Address,
         Code 0 (hop limit exceeded in transit),
         Interrupt packet processing and discard the packet.
S07.   }
S08.   max_last_entry = (Hdr Ext Len / 2) - 1
S09.   If ((Last Entry > max_last_entry) or
           (Segments Left > (Last Entry + 1))) {
S10.     Send an ICMP Parameter Problem message to the Source Address,
         Code 0 (Erroneous header field encountered),
         Pointer set to the Segments Left field,
         Interrupt packet processing and discard the packet.
S11.   }
S12.   Decrement Hop Limit by 1.
S13.   Decrement Segments Left by 1.
S14.   Copy Segment List[Segments Left] from the SRH to the
       Destination Address of the IPv6 header.
S15.   If (Upper-layer header type != 41 (IPv6)) {
S16.     Resubmit the packet to the IPv6 module for transmission to
         the new destination.
S17.   }
S18.   Perform IPv6 decapsulation.
S19.   Submit the packet to the IPv6 module for transmission on
       interface IFACE-OUT via NH-ADDR.
S20. }
```

Figure 18: SID processing for SRv6 static proxy (IPv6)

When processing the Upper-layer header of a packet matching a FIB entry locally instantiated as an SRv6 static proxy SID for IPv6 traffic, the following pseudocode is executed.

```
S01. If (Upper-layer header type != 41 (IPv6)) {
S02.   Process as per [RFC8986] Section 4.1.1
S03. }
S04. Perform IPv6 decapsulation.
S05. Submit the packet to the IPv6 module for transmission on
     interface IFACE-OUT via NH-ADDR.
```

Figure 19: Upper-layer header processing for SRv6 static proxy (IPv6)

When processing an IPv6 packet received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN, the following pseudocode is executed.

```
S01. Retrieve the CACHE entry associated with IFACE-IN.
S02. If the CACHE entry is not empty {
S03.   Decrement the Hop Limit.
S04.   Perform IPv6 encapsulation with an SRH
         Source Address of the IPv6 header is set to CACHE.SA,
         Destination Address of the IPv6 header is set to
         CACHE.LIST[0],
         Next Header of the SRH is set to 41 (IPv6),
         Segment List of the SRH is set to CACHE.LIST.
S05.   Submit the packet to the IPv6 module for transmission to the
       next destination.
S06. }
```

Figure 20: Inbound policy for SRv6 static proxy (IPv6)

S04: CACHE.LIST[0] represents the first entry in CACHE.LIST. Unless a local configuration indicates otherwise, the SIDs in CACHE.LIST should be encoded in the Segment List field in reversed order, the Segment Left and Last Entry values should be set of the length of CACHE.LIST minus 1. If CACHE.LIST contains a single entry, the SRH can be omitted and the Next Header field of the (outer) IPv6 header set to 41 (IPv6).

## 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 service. 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.

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 service.

The cache entry is not mapped to any particular packet, but instead to an SR service 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 service may thus drop, modify or generate new packets without affecting the proxy.

#### 6.2.1. SR-MPLS Pseudocode

The dynamic proxy SR-MPLS pseudocode is obtained by inserting the following instructions at the beginning of the static SR-MPLS pseudocode (Section 6.1.1).

```
S01. If the top label S bit is different from 0 {
S02.   Discard the packet.
S03. }
S04. POP the top label.
S05. Copy the MPLS label stack in a CACHE entry associated with the
     interface IFACE-IN.
```

Figure 21: SID processing for MPLS dynamic proxy

S01: As mentioned at the beginning of Section 6, an SR proxy is not needed to include an SR-unaware service at the end of an SR policy.

S05: An implementation may optimize the caching procedure by copying information into the cache only if it is different from the current content of the cache entry. Furthermore, 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).

When processing an Ethernet frame, an IPv4 packet or an IPv6 packet received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN, the pseudocode reported in Figure 7, Figure 9 or Figure 11, respectively, is executed.

#### 6.2.2. SRv6 Pseudocode

When processing an IPv6 packet matching a FIB entry locally instantiated as an SRv6 dynamic proxy SID, the same pseudocode as described in Figure 12, Figure 15 and Figure 18, respectively for Ethernet, IPv4 and IPv6 traffic, is executed with the following addition between lines S17 and S18.

```
(... S17.     })
S17.1.   Copy the IPv6 encapsulation in a CACHE entry associated with
         the interface IFACE-IN.
(S18.     Perform IPv6 decapsulation...)
```

Figure 22: SID processing for SRv6 dynamic proxy

An implementation may optimize the caching procedure by copying information into the cache only if it is different from the current content of the cache entry. A Hop Limit margin can be configured 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 Hop Limit difference smaller than the configured margin should not trigger a cache update. Similarly, the Flow Label value can be ignored when comparing the current packet IPv6 header with the cache entry. In this case, the Flow Label should be re-computed by the proxy node when it restores the IPv6 encapsulation from the cache entry.

When processing the Upper-layer header of a packet matching a FIB entry locally instantiated as an SRv6 dynamic proxy SID, process the packet as per [RFC8986] Section 4.1.1.

When processing an Ethernet frame, an IPv4 packet or an IPv6 packet received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN, the same pseudocode as in Figure 14, Figure 17 or Figure 20, respectively, is executed.

### 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 service. This proxy behavior leverages a shared-memory interface with a virtualized service (VNF) in order to hide the SR information from an SR-unaware service while keeping it attached to the packet. We assume in this case that the proxy and the VNF are running on the same compute node. A typical scenario is an SR-capable vrouter running on a container host and forwarding traffic to VNFs isolated within their respective container.

### 6.4. Masquerading SR Proxy

The masquerading proxy is an SR endpoint behavior for processing SRv6 traffic on behalf of an SR-unaware service. 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:

* NH-ADDR: Next hop Ethernet address

* IFACE-OUT: Local interface for sending traffic towards the service

* IFACE-IN: Local interface receiving the traffic coming back from the service

A masquerading SR proxy segment is thus defined for a specific service 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 service 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 service 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 SID. The proxy inspects the IPv6 extension headers and substitutes the Destination Address with the last SID 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 service.

The service 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 service simply forwards the packet based on its current Destination Address. In this scenario, we assume that the service 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 services that can be supported by a masquerading SR proxy. Flavors of the masquerading behavior are defined in Section 6.4.2 and Section 6.4.3 to support a wider range of services.

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 service, 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 SID as the Destination Address of the IPv6 packet.

#### 6.4.1. SRv6 Masquerading Proxy Pseudocode

Masquerading: When processing an IPv6 packet matching a FIB entry locally instantiated as an SRv6 masquerading proxy SID, the following pseudocode is executed.

```
S01. When an SRH is processed {
S02.   If (Segments Left == 0) {
S03.     Proceed to process the next header in the packet.
S04.   }
S05.   If (IPv6 Hop Limit <= 1) {
S06.     Send an ICMP Time Exceeded message to the Source Address,
         Code 0 (hop limit exceeded in transit),
         Interrupt packet processing and discard the packet.
S07.   }
S08.   max_last_entry = (Hdr Ext Len / 2) - 1
S09.   If ((Last Entry > max_last_entry) or
           (Segments Left > (Last Entry + 1))) {
S10.     Send an ICMP Parameter Problem message to the Source Address,
         Code 0 (Erroneous header field encountered),
         Pointer set to the Segments Left field,
         Interrupt packet processing and discard the packet.
S11.   }
S12.   Decrement Hop Limit by 1.
S13.   Decrement Segments Left by 1.
S14.   Copy Segment List[0] from the SRH to the Destination Address
       of the IPv6 header.
S15.   Submit the packet to the IPv6 module for transmission on
       interface IFACE-OUT via NH-ADDR.
S16. }
```

Figure 23: SID processing for SRv6 masquerading proxy

When processing the Upper-layer header of a packet matching a FIB entry locally instantiated as an SRv6 masquerading proxy SID, process the packet as per [RFC8986] Section 4.1.1.

De-masquerading: When processing an IPv6 packet received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN, the following pseudocode is executed.

```
S01. When an SRH is processed {
S02.   If (IPv6 Hop Limit <= 1) {
S03.     Send an ICMP Time Exceeded message to the Source Address,
         Code 0 (hop limit exceeded in transit),
         Interrupt packet processing and discard the packet.
S04.   }
S05.   If (Segments Left != 0) {
S06.     max_last_entry = (Hdr Ext Len / 2) - 1
S07.     If ((Last Entry > max_last_entry) or
             (Segments Left > Last Entry)) {
S08.       Send an ICMP Parameter Problem message to the Source Address,
           Code 0 (Erroneous header field encountered),
           Pointer set to the Segments Left field,
           Interrupt packet processing and discard the packet.
S09.     }
S10.     Copy Segment List[Segments Left] from the SRH to the
         Destination Address of the IPv6 header.
S11.   }
S12.   Decrement Hop Limit by 1.
S13.   Submit the packet to the IPv6 module for transmission to the
       next destination.
S14. }
```

Figure 24: Inbound policy for SRv6 masquerading proxy

### 6.4.2. Destination NAT Flavor

Services modifying the destination address in the packets they process, such as NATs, can be supported by reporting the updated Destination Address back into the Segment List field of the SRH.

The Destination NAT flavor of the SRv6 masquerading proxy is enabled by adding the following instruction between lines S09 and S10 of the de-masquerading pseudocode in Figure 24.

```
(... S09.     })
S09.1.   Copy the Destination Address of the IPv6 header to the
         Segment List[0] entry of the SRH.
(S10.     Copy Segment List[Segments Left] from the SRH to the
          Destination Address of the IPv6 header...)
```

### 6.4.3. Caching Flavor

Services 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.

The caching flavor of the SRv6 masquerading proxy is enabled by:

* Adding the following instruction between lines S14 and S15 of the masquerading pseudocode in Figure 23.

```
(... S14.   Copy Segment List[0] from the SRH to the Destination
            Address of the IPv6 header.
S14.1. Copy the IPv6 encapsulation in a CACHE entry associated with
       the interface IFACE-IN.
(S15.   Submit the packet to the IPv6 module for transmission on
        interface IFACE-OUT via NH-ADDR.)
```

* Updating the de-masquerading pseudocode such that, in addition to the SRH processing in Figure 24, the following pseudocode is executed when processing an IPv6 packet (received on the interface IFACE-IN and with a destination address that does not match any address of IFACE-IN) that does not contain an SRH.

```
S01. Retrieve the CACHE entry associated with IFACE-IN.
S02. If the CACHE entry is not empty {
S03.   If (IPv6 Hop Limit <= 1) {
S04.     Send an ICMP Time Exceeded message to the Source Address,
         Code 0 (hop limit exceeded in transit),
         Interrupt packet processing and discard the packet.
S05.   }
S06.   Decrement Hop Limit by 1.
S07.   Update the IPv6 encapsulation according to the retrieved CACHE
       entry.
S08.   Submit the packet to the IPv6 module for transmission to the
       next destination.
S09. }
```

# 7. Metadata

## 7.1. MPLS Data Plane

Metadata can be carried for SR-MPLS traffic in a Segment Routing Header inserted between the last MPLS label and the MPLS payload. When used solely as a metadata container, the SRH does not carry any segment but only the mandatory header fields, including the tag and flags, and any TLVs that is required for transporting the metadata.

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 in an MPLS packet is a potential issue to be addressed. One possible solution is to add the indication about the presence of metadata in the semantic of the SIDs. Note that only the SIDs whose behavior involves looking at the metadata or the MPLS payload would need to include such semantic (e.g., service segments). Other segments, such as topological

segments, are not affected by the presence of metadata. Another, more generic, solution is to introduce a protocol identifier field within the MPLS packet as described in [I-D.xu-mpls-payload-protocol-identifier].

## 7.2. IPv6 Data Plane

### 7.2.1. SRH TLV Objects

The IPv6 SRH TLV objects are designed to carry all sorts of metadata. TLV objects can be imposed by the ingress edge router that steers the traffic into the SR service policy.

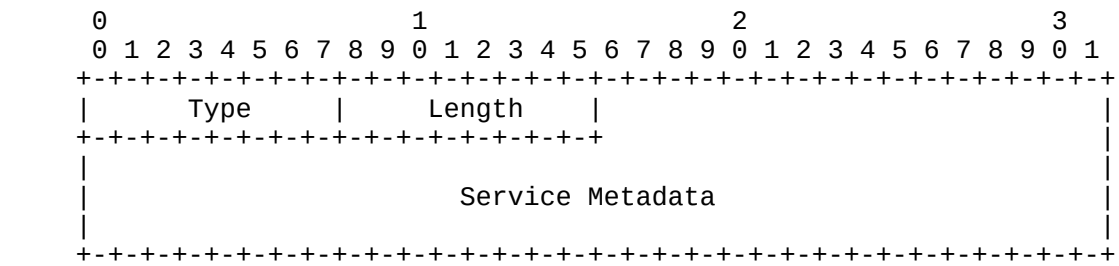
An SR-aware service may impose, modify or remove any TLV object attached to the first SRH, either by directly modifying the packet headers or via a control channel between the service and its forwarding plane.

An SR-aware service that re-classifies the traffic and steers it into a new SR service 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.

#### 7.2.1.1. Opaque Metadata TLV

This document defines an SRv6 TLV called Opaque Metadata TLV. This is a fixed-length container to carry any type of Service Metadata. No assumption is made by this document on the structure or the content of the carried metadata. The Opaque Metadata TLV has the following format:




where:

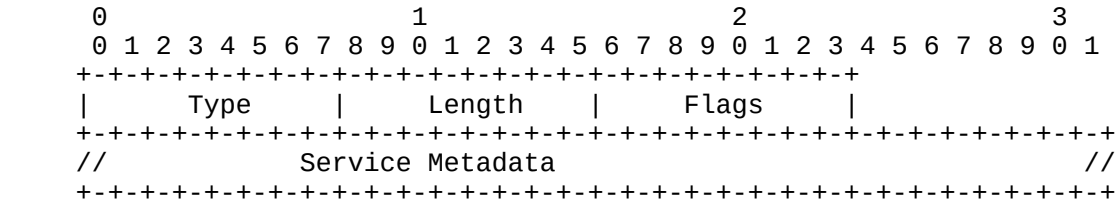
* Type: to be assigned by IANA.

* Length: 14.

* Service Metadata: 14 octets of opaque data.

#### 7.2.1.2. NSH Carrier TLV

This document defines an SRv6 TLV called NSH Carrier TLV. It is a container to carry Service Metadata in the form of Variable-Length Metadata as defined in [RFC8300] for NSH MD Type 2. The NSH Carrier TLV has the following format:




where:

* Type: to be assigned by IANA.

* Length: the total length of the TLV.

* Flags: 8 bits. No flags are defined in this document. SHOULD be set to 0 on transmission and MUST be ignored on receipt.

* Service Metadata: a list of Service Metadata TLV as defined in [RFC8300] for NSH MD Type 2.

### 7.2.2. SRH Tag

The SRH tag identifies a packet as part of a group or class of packets [RFC8754].

In the context of service programming, this field can be used to encode basic metadata in the SRH. An example use-case is to leverage the SRH tag to encode a policy ID. This policy ID can then be used by an SR-aware function to identify a particular processing policy to be applied on that packet.

## 8. Implementation Status

This section is to be removed prior to publishing as an RFC.

### 8.1. SR-Aware Services

Specific SRv6 support has been implemented for the below open-source services:

* Iptables (1.6.2 and later) [IPTABLES]

* Nftables (0.8.4 and later) [NFTABLES]

* Snort [SNORT]

In addition, any service relying on the Linux kernel, version 4.10 and later, or FD.io VPP for packet forwarding can be considered as SR-aware.

## 8.2. Proxy Behaviors

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 | Available | Available |
| | Shared memory proxy | In progress | In progress |
| | Masquerading proxy | Available | Available |

Figure 25: Open-source implementation status table

# 9. Related Works

The Segment Routing solution addresses a wide problem that covers both topological and service 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 [RFC8300] and the MPLS label swapping mechanism

described in [RFC8595], which rely on state configured at every hop of the service chain.

## 10. IANA Considerations

### 10.1. SRv6 Endpoint Behaviors

This I-D requests the IANA to allocate, within the "SRv6 Endpoint Behaviors" sub-registry belonging to the top-level "Segment-routing with IPv6 dataplane (SRv6) Parameters" registry, the following allocations:

| Value | Description | Reference |
|---|---|---|
| TBA1-1 | End.AN - SR-aware function (native) | [This.ID] |
| TBA1-2 | End.AS - Static proxy | [This.ID] |
| TBA1-3 | End.AD - Dynamic proxy | [This.ID] |
| TBA1-4 | End.AM - Masquerading proxy | [This.ID] |
| TBA1-5 | End.AM - Masquerading proxy with NAT | [This.ID] |
| TBA1-6 | End.AM - Masquerading proxy with Caching | [This.ID] |
| TBA1-7 | End.AM - Masquerading proxy with NAT & Caching | [This.ID] |

### 10.2. Segment Routing Header TLVs

This I-D requests the IANA to allocate, within the "Segment Routing Header TLVs" registry, the following allocations:

| Value | Description | Reference |
|---|---|---|
| TBA2-1 | Opaque Metadata TLV | [This.ID] |
| TBA2-2 | NSH Carrier TLV | [This.ID] |

## 11. Security Considerations

The security requirements and mechanisms described in [RFC8402], [RFC8754] and [RFC8986] also apply to this document.

This document does not introduce any new security vulnerabilities.

## 12. Acknowledgements

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

## 13. References

### 13.1. Normative References

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

[RFC8660] Bashandy, A., Ed., Filsfils, C., Ed., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with the MPLS Data Plane", RFC 8660, DOI 10.17487/RFC8660, December 2019, <https://www.rfc-editor.org/info/rfc8660>.

[RFC8754] Filsfils, C., Ed., Dukes, D., Ed., Previdi, S., Leddy, J., Matsushima, S., and D. Voyer, "IPv6 Segment Routing Header (SRH)", RFC 8754, DOI 10.17487/RFC8754, March 2020, <https://www.rfc-editor.org/info/rfc8754>.

[RFC8986] Filsfils, C., Ed., Camarillo, P., Ed., Leddy, J., Voyer, D., Matsushima, S., and Z. Li, "Segment Routing over IPv6 (SRv6) Network Programming", RFC 8986, DOI 10.17487/RFC8986, February 2021, <https://www.rfc-editor.org/info/rfc8986>.

[RFC9256] Filsfils, C., Talaulikar, K., Ed., Voyer, D., Bogdanov, A., and P. Mattes, "Segment Routing Policy Architecture", RFC 9256, DOI 10.17487/RFC9256, July 2022, <https://www.rfc-editor.org/info/rfc9256>.

### 13.2. Informative References

[I-D.dawra-idr-bgp-sr-service-chaining] Dawra, G., Filsfils, C., Bernier, D., Uttaro, J., Decraene, B., Elmalky, H., Xu, X., Clad, F., and K. Talaulikar, "BGP Control Plane Extensions for Segment Routing based Service Chaining", Work in Progress, Internet-Draft, draft-dawra-idr-bgp-sr-service-chaining-02, 4 January 2018, <https://datatracker.ietf.org/doc/html/draft-dawra-idr-bgp-sr-service-chaining-02>.

[I-D.xu-mpls-payload-protocol-identifier] Xu, X., Assarpour, H., Ma, S., and F. Clad, "MPLS Payload Protocol Identifier", Work in Progress, Internet-Draft, draft-xu-mpls-payload-protocol-identifier-09, 2 September 2021, <https://datatracker.ietf.org/doc/html/draft-xu-mpls-payload-protocol-identifier-09>.

[IFIP18] Abdelsalam, A., Salsano, S., Clad, F., Camarillo, P., and C. Filsfils, "SEgment Routing Aware Firewall For Service Function Chaining scenarios", IFIP Networking conference , May 2018.

[IPTABLES] "iptables-1.6.2 changes", February 2018, <https://netfilter.org/projects/iptables/files/changes-iptables-1.6.2.txt>.

[NFTABLES] "nftables-0.8.4 changes", May 2018, <https://netfilter.org/projects/nftables/files/changes-nftables-0.8.4.txt>.

[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>.

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

[RFC8595] Farrel, A., Bryant, S., and J. Drake, "An MPLS-Based Forwarding Plane for Service Function Chaining", RFC 8595, DOI 10.17487/RFC8595, June 2019, <https://www.rfc-editor.org/info/rfc8595>.

[SNORT] "SR-Snort", March 2018, <https://github.com/SRouting/SR-Snort>.

## Contributors

Pablo Camarillo
Cisco Systems, Inc.
Spain
Email: pcamaril@cisco.com

Bart Peirens
Proximus
Belgium
Email: bart.peirens@proximus.com

Dirk Steinberg
Lapishills Consulting Limited
Cyprus

Email: dirk@lapishills.com

Ahmed AbdelSalam
Cisco Systems, Inc.
Italy
Email: ahabdels@cisco.com

Gaurav Dawra
LinkedIn
United States of America
Email: gdawra@linkedin.com

Stewart Bryant
Futurewei Technologies Inc
Email: stewart.bryant@gmail.com

Hamid Assarpour
Broadcom
Email: hamid.assarpour@broadcom.com

Himanshu Shah
Ciena
Email: hshah@ciena.com

Luis M. Contreras
Telefonica I+D
Spain
Email: luismiguel.contrerasmurillo@telefonica.com

Jeff Tantsura
Individual
Email: jefftant@gmail.com

Martin Vigoureux
Nokia
Email: martin.vigoureux@nokia.com

Jisu Bhattacharya
Cisco Systems, Inc.
United States of America
Email: jisu@cisco.com

## Authors' Addresses

Francois Clad (editor)
Cisco Systems, Inc.
France
Email: fclad.ietf@gmail.com

Xiaohu Xu (editor)
China Mobile
Email: xuxiaohu@cmss.chinamobile.com

Clarence Filsfils
Cisco Systems, Inc.
Belgium
Email: cf@cisco.com

Daniel Bernier
Bell Canada
Canada
Email: daniel.bernier@bell.ca

Cheng Li
Huawei
Email: chengli13@huawei.com

Bruno Decraene
Orange
France
Email: bruno.decraene@orange.com

Shaowen Ma
Mellanox
Email: mashaowen@gmail.com

Chaitanya Yadlapalli
AT&T
United States of America
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