



Nokia Service Router Linux

MPLS Guide Release 21.6

3HE 17820 AAAA TQZZA

Edition: 1

September 2021

© 2021 Nokia.

Use subject to Terms available at: www.nokia.com

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

This document includes Nokia proprietary and confidential information, which may not be distributed or disclosed to any third parties without the prior written consent of Nokia.

This document is intended for use by Nokia's customers ("You"/"Your") in connection with a product purchased or licensed from any company within Nokia Group of Companies. Use this document as agreed. You agree to notify Nokia of any errors you may find in this document; however, should you elect to use this document for any purpose(s) for which it is not intended, You understand and warrant that any determinations You may make or actions You may take will be based upon Your independent judgment and analysis of the content of this document.

Nokia reserves the right to make changes to this document without notice. At all times, the controlling version is the one available on Nokia's site.

No part of this document may be modified.

NO WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF AVAILABILITY, ACCURACY, RELIABILITY, TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, IS MADE IN RELATION TO THE CONTENT OF THIS DOCUMENT. IN NO EVENT WILL NOKIA BE LIABLE FOR ANY DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL OR ANY LOSSES, SUCH AS BUT NOT LIMITED TO LOSS OF PROFIT, REVENUE, BUSINESS INTERRUPTION, BUSINESS OPPORTUNITY OR DATA THAT MAY ARISE FROM THE USE OF THIS DOCUMENT OR THE INFORMATION IN IT, EVEN IN THE CASE OF ERRORS IN OR OMISSIONS FROM THIS DOCUMENT OR ITS CONTENT.

Copyright and trademark: Nokia is a registered trademark of Nokia Corporation. Other product names mentioned in this document may be trademarks of their respective owners.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

© 2021 Nokia.

Table of contents

1	Getting started	5
1.1	About this document.....	5
1.2	What's new	5
1.3	Precautionary messages	5
1.4	Conventions.....	6
2	Overview	7
2.1	About MPLS	7
2.1.1	LSRs.....	7
2.2	About LDP	8
2.3	Supported functionality	9
3	Configuring MPLS	11
3.1	MPLS label manager	11
3.1.1	Static and dynamic label blocks	11
3.1.2	Configuring label blocks	12
3.1.3	Displaying label block information	12
3.2	Static MPLS forwarding	13
3.2.1	Configuring an ingress LER.....	14
3.2.2	Configuring a transit LSR	14
3.2.3	Configuring PHP.....	15
3.3	ACLs and MPLS traffic	16
3.4	MPLS MTU	16
3.4.1	Configuring the MPLS MTU.....	17
3.4.2	Displaying MPLS MTU information.....	18
3.5	TTL propagation and TTL expiry	18
3.6	MPLS traffic classification and marking	18
3.6.1	Ingress LER.....	19
3.6.2	Transit LSR.....	20
3.6.3	PHP LSR	20
3.6.4	Egress LER	21
3.6.5	Default MPLS Traffic-Class Classifier Policy	22
3.7	Show reports for MPLS tunnel tables	22
4	Configuring LDP	25
4.1	Enabling LDP.....	25
4.2	Configuring LDP neighbor discovery	25
4.3	Configuring LDP peers	26
4.4	Configuring a label block for LDP	27
4.5	Configuring longest-prefix match for IPv4 FEC resolution.....	28
4.6	Configuring load balancing over equal-cost paths.....	28
4.7	Configuring graceful restart helper capability	29
4.8	Configuring LDP-IGP synchronization	30

1 Getting started

This chapter provides an overview of this document, includes summaries of changes from previous releases, and lists precautionary messages and command conventions.

1.1 About this document

This guide describes the services and provides configuration examples for the Multiprotocol Label Switching (MPLS) protocol used with the Nokia Service Router Linux (SR Linux).

This document is intended for users who plan to implement MPLS for SR Linux.



Note: This manual covers the current release and may also contain some content that will be released in later maintenance loads. Refer to the *SR Linux Release Notes* for information on features supported in each load.

1.2 What's new





This is the first release of this document.

1.3 Precautionary messages

Observe all dangers, warnings, and cautions in this document to avoid injury or equipment damage during installation and maintenance. Follow the safety procedures and guidelines when working with and near electrical equipment.

[Table 1](#) describes information symbols contained in this document.

Table 1 Information symbols

Symbol	Meaning	Description
	Danger	Warns that incorrect handling and installation could result in bodily injury. An electric shock hazard could exist. Before beginning work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.
	Warning	Warns that incorrect handling and installation could result in equipment damage or loss of data.
	Caution	Warns that incorrect handling may reduce component or system performance.
	Note	Notes contain suggestions or additional operational information.

1.4 Conventions

Nokia SR Linux documentation uses the following command conventions.

- **Bold** type indicates a command that the user must enter.
- Input and output examples are displayed in `Courier` text.
- An open right angle bracket indicates a progression of menu choices or simple command sequence (often selected from a user interface). Example: **start > connect to**
- Angle brackets (< >) indicate an item that is not used verbatim. For example, for the command **show ethernet <name>**, *name* should be replaced with the name of the interface.
- A vertical bar (|) indicates a mutually exclusive argument.
- Square brackets ([]) indicate optional elements.
- Braces ({ }) indicate a required choice. When braces are contained within square brackets, they indicate a required choice within an optional element.
- *Italic* type indicates a variable.

Generic IP addresses are used in examples. Replace these with the appropriate IP addresses used in the system.

2 Overview

This chapter provides a brief description of MPLS and LDP and lists the functionality supported by SR Linux in this release.

2.1 About MPLS

Multiprotocol Label Switching (MPLS) is a label switching technology that provides the ability to set up connection-oriented paths over a connectionless IP network. MPLS facilitates network traffic flow and provides a mechanism to engineer network traffic patterns independently from routing tables. MPLS sets up a specific path for a sequence of packets. The packets are identified by a label stack inserted into each packet.

MPLS requires a set of procedures to enhance network-layer packets with label stacks, which then turns them into labeled packets. Routers that support MPLS are known as Label Switching Routers (LSRs). To transmit a labeled packet on a particular data link, an LSR must support the encoding technique.

In MPLS, packets can carry not only one label but a set of labels in a stack. An LSR can swap the label at the top of the stack, pop the label, or swap the label and push one or more labels into the stack. The processing of a labeled packet is completely independent of the level of hierarchy. The processing is always based on the top label, without regard for the possibility that some number of other labels may have been above it in the past, or that some number of other labels may be below it at present.

2.1.1 LSRs

LSRs perform different label switching functions based on their position in a Label Switched Path (LSP). The LSRs in an LSP do one of the following:

- The LSR at the or head-end of an LSP is the ingress label edge router (LER). The ingress LER can encapsulate packets with an MPLS header and forward them to the next router along the path. A point-to-point LSP can only have one ingress LER.

The ingress LER is programmed to perform a label push operation. More specifically, it is programmed with an IP route that encapsulates matching IP packets using MPLS and forwards them to one or more next-hop routers. Each outgoing MPLS packet has a label stack (in the MPLS header), and the top entry in each stack contains a label value pushed by the route lookup process that indicates the path to be followed.

- A transit LSR is an intermediate router in the LSP between the ingress and egress routers. Each transit LSR along the path is programmed to perform a label swap operation: the transit LSR is programmed with an MPLS route that matches the label value at the top of the label stack, pops that label stack entry, pushes a new top label and forwards the resulting MPLS packet to the next set of routers along the path.
- The penultimate LSR (the one before last) in the LSP can be programmed to perform a pop-and-forward operation, known as penultimate hop popping (PHP). In this case, the LSR is programmed with an MPLS route that matches the label value at the top of the label stack, pops that label stack entry, and forwards the resulting packet to the tail-end router. The packet sent by the PHP LSR to the egress LER may be the original IP payload packet, but the forwarding decision made by the PHP LSR is based only on the incoming top label value, not IP route lookup.
- The router at the tail-end of an LSP is the egress LER. The egress LER strips the MPLS encapsulation, which changes it from an MPLS packet to a data packet, and then forwards the packet to its destination using information in the forwarding table. Each point-to-point LSP can have only one egress router. The ingress and egress routers in an LSP cannot be the same router.

If the penultimate LSR is just a normal transit LSR performing a label swap (no PHP), the egress LER must be programmed to perform the pop operation. In this case, the programmed MPLS route matches the label value at the top of the label stack, the egress LER pops that label entry and then does another lookup on the next header; usually this is an IP header and the packet is forwarded based on IP route lookup.

2.2 About LDP

Label Distribution Protocol (LDP) is a protocol used to distribute MPLS labels in non-traffic-engineered applications. LDP allows routers to establish LSPs through a network by mapping network-layer routing information directly to data link layer-switched paths.

An LSP is defined by the set of labels from the ingress LER to the egress LER. LDP associates a Forwarding Equivalence Class (FEC) with each LSP it establishes. A FEC is a collection of common actions associated with a class of packets. When an LSR assigns a label to a FEC, it must allow other LSRs in the path to know about the label. LDP helps to establish the LSP by providing a set of procedures that LSRs can use to distribute labels.

The FEC associated with an LSP specifies which packets are mapped to that LSP. LSPs are extended through a network as each LSR splices incoming labels for a FEC to the outgoing label assigned to the next-hop for the specific FEC.

LDP performs the label distribution only in MPLS environments. The LDP operation begins with a hello discovery process to find LDP peers in the network. LDP peers are two LSRs that use LDP to exchange label/FEC mapping information. An LDP session is created between LDP peers. A single LDP session allows each peer to learn the other's label mappings (LDP is bidirectional) and to exchange label binding information.

LDP signaling works with the MPLS label manager to manage the relationships between labels and the corresponding FEC. An MPLS label identifies a set of actions that the forwarding plane performs on an incoming packet before discarding it. The FEC is identified through the signaling protocol (in this case, LDP) and allocated a label. The mapping between the label and the FEC is communicated to the forwarding plane.

When an unlabeled packet ingresses the router, classification policies associate it with a FEC. The appropriate label is imposed on the packet, and the packet is forwarded.

2.3 Supported functionality

In the current release, SR Linux supports the following MPLS and LDP functionality:

MPLS

- Statically configured MPLS forwarding entries
- Configurable label range
- MPLS label manager that shares the MPLS label space among client applications that require MPLS labels

LDP

- LDPv4 implementation compliant with RFC 5036

-
- LDP support in the default network-instance only
 - Label distribution using DU (downstream unsolicited), ordered control
 - Platform label space only
 - Configurable label range (dynamic, non-shared label-block)
 - Support for overload TLV when label-block has no free entries
 - Configurable timers (hello-interval, hello-holdtime, keepalive-interval)
 - Ingress LER, transit LSR, and egress LER roles for /32 IPv4 FECs
 - Automatic FEC origination of the system0.0 /32 IPv4 address prefix
 - /32 IPv4 FEC resolution using IGP routes, with longest prefix match option
 - ECMP support with configurable next-hop limit (up to 64)
 - Automatic installation of all LDP /32 IPv4 prefix FECs into TTM
 - Per-peer configurable FEC limit
 - Graceful restart helper capability
 - BGP shortcuts for IPv4 traffic
 - Protocol debug/trace-options
 - LDP-IGP synchronization
 - Advertise address mapping message for primary IPv4 address of the adjacent interface only.
 - Non-configurable capability advertisement in INITIALIZATION messages only claiming support for:
 - State advertisement control (SAC) with interest in IPv4 prefix FECs only
 - Fault tolerance (Graceful Restart)
 - Nokia overload TLV
 - Unrecognized notification
 - Split-horizon support: A label-mapping message is not advertised to a peer if the FEC matches an address sent by that peer in an Address Mapping message.

3 Configuring MPLS

This chapter provides information about how MPLS functions on SR Linux and examples of common configuration tasks. It contains the following topics:

- [MPLS label manager](#)
- [Static MPLS forwarding](#)
- [ACLs and MPLS traffic](#)
- [MPLS MTU](#)
- [TTL propagation and TTL expiry](#)
- [MPLS traffic classification and marking](#)
- [Show reports for MPLS tunnel tables](#)

3.1 MPLS label manager

SR Linux features an MPLS label manager process that shares the MPLS label space among client applications that require MPLS labels; these applications include static MPLS forwarding and LDP.

For a protocol such as LDP to become operational, it must be configured with a reference to a predefined range of labels, called a label block. A label block configuration includes a start-label value and an end-label value. When the label block is made available to an application, the application can use any label in the range between the start-label and end-label, inclusive of those values.

The MPLS label manager ensures there is no configuration overlap between the labels of two different label blocks.

3.1.1 Static and dynamic label blocks

A label block can be static or dynamic:

- A static label block is provided by the MPLS label manager to a client application when it is expected that the client application will specify the exact label value it wants to use with every label request.
- A dynamic label block is provided by the MPLS label manager to a client application when it is expected that the client application will request the next available label when it needs a new entry.

A label block can be configured as shared or dedicated. When a label block is configured to be shared, it allows the label block to be made available to multiple protocols. If a label block is not configured as shared, it is reserved for the exclusive use of one protocol.

The label block used by LDP must be configured as a dynamic, non-shared label block. For static MPLS routes, it is necessary to configure a static label block and reference the static label block in the static MPLS configuration.

3.1.2 Configuring label blocks

The following example configures a static and dynamic label block.

Example:

```
--{ * candidate shared default }--[ ]--
# info system mpls
  system {
    mpls {
      label-ranges {
        static s1 {
          start-label 10001
          end-label 11000
        }
        dynamic d1 {
          start-label 11001
          end-label 12000
        }
      }
    }
  }
}
```

3.1.3 Displaying label block information

Use the **info from state** command to display information about the configured label blocks.

Example:

```
--{ * candidate shared default }--[ ]--
# info from state system mpls label-ranges
  system {
    mpls {
      label-ranges {
        static {
          name s1
          start-label 10001
        }
      }
    }
  }
}
```

```

        end-label 11000
        allocated-labels 10
        free-labels 990
        status ready
    }
}
}
}
}
}

```

The status for a label block can be one of the following:

- **ready** – The client application has full access to all the labels from the configured start-label to the configured end-label, without restriction.
- **not-ready** – The client application has no access to the labels in the label range because initial allocation is still in process and the MPLS label manager may be waiting on other blocks to clean up first.
- **delete-pending** – The label block has been deleted from the running configuration but it still shows in `info from state` output with this status until all the labels are released.
- **updating** – When a label block is resized the status is shown as updating; during this time the client application has access only to the labels from the previous start-label to the previous end-label (which were available before the latest transaction), and this operational label-range is shown by the **info from state** values for `start-label`, `end-label`, and `free-labels`.

3.2 Static MPLS forwarding

Statically configured MPLS forwarding entries allow MPLS-labeled packets to be forwarded along a specific path that may differ from the normal shortest path provided by the underlay routing protocol.

Static next-hop-groups used by IPv4 and IPv6 static routes of the default network-instance support MPLS next hops. An MPLS next-hop has a next-hop IP address and a list of MPLS labels (currently limited to 1). When an IP packet matches a static route pointing to a next-hop-group with MPLS next-hops, the packet is MPLS encapsulated according to the selected next-hop.

Static MPLS routes are supported in the default network-instance. Static MPLS routes control the way that transit LSRs, penultimate LSRs, and egress LERs handle received MPLS packets. Each static MPLS route matches a particular label value and causes that label value to be popped from the label stack when it appears as the top label in any received MPLS packet, on any interface. If the static MPLS route points to a next-hop-group with MPLS next-hops, the packet is forwarded to the

selected next-hop with a swap operation; ECMP is supported if there are multiple MPLS next-hops. When the `pushed-mpls-label-stack` parameter for the MPLS next-hop specifies the IPv4 or IPv6 `IMPLICIT_NULL` label value, no new label is pushed and a PHP pop-and-forward operation is performed.

3.2.1 Configuring an ingress LER

The following example shows the default network-instance configured for an ingress LER, specifying the label to push to MPLS-encapsulated packets. In this example, multiple next-hops (ECMP) are specified in a next-hop group.

Example:

```
--{ * candidate shared default }--[ ]--
# info network-instance default next-hop-groups group srva_tora_ipv4
  network-instance default {
    next-hop-groups {
      group srva_tora_ipv4 {
        nexthop 0 {
          ip-address 192.13.1.3
          resolve false
          pushed-mpls-label-stack [
            544334
          ]
        }
        nexthop 2 {
          ip-address 192.13.1.3
          resolve false
          pushed-mpls-label-stack [
            205679
          ]
        }
      }
    }
  }
}
```

3.2.2 Configuring a transit LSR

The following example shows the default network-instance configured as a transit LSR in the label-switched path.

Example:

```
--{ * candidate shared default }--[ ]--
# info network-instance default mpls
  network-instance default {
    mpls {
      static-label-block s2
    }
  }
}
```

```

        static-entry 1000 preference 100 {
            operation swap
            next-hop-group nhop_group_1
        }
    }
}

```

This example configures MPLS to use a static label block named `s2`. The static label block is configured with a start-label value and an end-label value. See [Configuring label blocks](#) for an example of a static label block configuration.

The example configures incoming MPLS transit traffic with label 1000 to use the next-hops in next-hop-group `nhop_group_1` for label-swap operations.

Example:

The following example specifies the MPLS next-hop for `nhop_group_1`. Only one MPLS next-hop is supported per next-hop-group. In this example, the label for outgoing traffic to MPLS next-hop 192.35.1.5 is swapped to 1001.

```

--{ * candidate shared default }--[ ]--
# info network-instance default next-hop-groups group nhop_group_1
  network-instance default {
    next-hop-groups {
      group nhop_group_1 {
        nexthop 0 {
          ip-address 192.35.1.5
          resolve false
          pushed-mpls-label-stack [
            1001
          ]
        }
      }
    }
  }
}

```

3.2.3 Configuring PHP

The following example shows the default network-instance configured to perform PHP for the LSR. In this example, the setting for `pushed-mpls-label-stack` is `IMPLICIT_NULL`, which causes the label for outgoing traffic to MPLS next-hop 192.35.1.1 to be popped.

Example:

```

--{ * candidate shared default }--[ ]--
# info network-instance default next-hop-groups group nhop_group_2
  network-instance default {
    next-hop-groups {
      group nhop_group_2 {

```

```

        nexthop 0 {
            ip-address 192.35.1.1
            resolve false
            pushed-mpls-label-stack [
                IMPLICIT_NULL
            ]
        }
    }
}

```

3.3 ACLs and MPLS traffic

[Table 2](#) lists how traffic along an MPLS datapath is evaluated by each type of ACL that can be configured on SR Linux. See the *SR Linux Configuration Basics Guide* for information about configuring ACLs on SR Linux.

Table 2 How MPLS traffic is handled by SR Linux ACLs

MPLS datapath	Evaluated by ingress ACL rules?	Evaluated by egress ACL rules?	Evaluated by CPU QoS entries?	Evaluated by IP CPM filter rules?	Evaluated by IP capture filter rules?
IP → MPLS (LER)	Yes	No	N/A	N/A	Yes
MPLS → MPLS (LSR)	Yes	No	N/A	N/A	No
MPLS → term (label TTL expiry)	Yes	N/A	No	No	No
MPLS → IP (PHP)	Yes	Yes	N/A	N/A	No
MPLS → IP (UHP)	Yes	Yes	N/A	N/A	Yes
MPLS → IP → term (IP address is local)	Yes	N/A	Yes	Yes	Yes

3.4 MPLS MTU

The MPLS MTU defines the maximum-sized MPLS packet that can be transmitted from a routed subinterface, including the size of the transmitted label stack (4 bytes * number of label stack entries). If an MPLS packet containing any payload exceeds this size, the packet is dropped.

3.4.1 Configuring the MPLS MTU

SR Linux supports a system-wide default MPLS MTU value, which can be configured with a range of 1284-9496 bytes; default is 1508 bytes. In addition, you can configure a subinterface-level MPLS MTU, which applies to subinterfaces of type **routed**. The supported range for the subinterface-level MPLS MTU is 1284-9496 bytes. If no MPLS MTU is configured for a subinterface, the default MTU value is taken from the system-wide default MPLS MTU.

Each 7250 IXR IMM supports a maximum of four different MPLS MTU values. If a line card already has four different MPLS MTU values, and a fifth MPLS MTU value is configured for a subinterface on the same line card, the subinterface is brought down with the reason `mpls-mtu-resource-exceeded`.

If a subinterface has a configured (or inherited) MPLS MTU of x bytes and it is associated with an untagged port with port MTU y bytes, and $x+14 > y$ then the subinterface is brought down for reason `mpls-mtu-too-large`.

If a subinterface has a configured (or inherited) MPLS MTU of x bytes and it is associated with a dot1q port with port MTU y bytes, and $x+18 > y$ then the subinterface is brought down for reason `mpls-mtu-too-large`.

Examples:

The following example configures a system-wide default MPLS MTU value:

```
--{ * candidate shared default }--[ ]--
# info system mtu
  system {
    mtu {
      default-mpls-mtu 4096
    }
  }
```

The following example configures an MPLS MTU for a routed subinterface:

```
--{ * candidate shared default }--[ ]--
# info interface ethernet-1/1 subinterface 1
  interface ethernet-1/1 {
    subinterface 1 {
      type routed
      admin-state enable
      mpls-mtu 4096
      ipv4 {
        address 192.168.11.1/30 {
        }
      }
      ipv6 {
        address 2001:1::192:168:11:1/126 {
        }
      }
    }
  }
```

```
    }
}
```

3.4.2 Displaying MPLS MTU information

Use the **info from state** command to display MPLS MTU information.

Examples:

To display the system-wide default MPLS MTU value:

```
--{ * candidate shared default }--[ ]--
# info from state system mtu default-mpls-mtu
  system {
    mtu {
      default-mpls-mtu 4096
    }
  }
```

To display the MPLS MTU for a subinterface:

```
--{ * candidate shared default }--[ ]--
# info from state interface ethernet-1/1 subinterface 1 mpls-mtu
  interface ethernet-1/1 {
    subinterface 1 {
      mpls-mtu 4096
    }
  }
```

3.5 TTL propagation and TTL expiry

SR Linux supports the Uniform model for TTL propagation as described RFC 3032 and RFC 3443. The Uniform model makes all the nodes that an LSP traverses visible to nodes outside the tunnel.

3.6 MPLS traffic classification and marking

SR Linux supports EXP-inferred LSPs as described in RFC 3270. This allows multiple classes of service to be transported by a single LSP, with the EXP marking of each packet determining the correct per-hop behavior (PHB) to apply to each router.

On SR Linux, the mapping between an EXP value and a PHB is provided by an MPLS traffic-class classifier policy. A single router can have one or more of these policies so that some subinterfaces can have one policy applied and other subinterfaces can have another policy applied. Each traffic-class classifier policy consists of multiple mapping entries, each of which maps one unique EXP value to a (forwarding-class, drop-probability) tuple.

SR Linux also supports MPLS traffic-class rewrite policies. If MPLS-encapsulated packets are transmitted out an egress subinterface with such a policy bound to it, the EXP field in all pushed labels of these packets is based on the mapping rules of the policy. MPLS traffic-class rewrite rules associate a forwarding-class or a (forwarding-class, drop-probability) tuple with an EXP rewrite value.

SR Linux does not support the short-pipe model of RFC 3270.

3.6.1 Ingress LER

When an SR Linux router, acting as ingress LER, matches an IP packet to an LDP tunnel or a static MPLS forwarding entry:

- The ingress LER determines the forwarding-class and drop-probability of the packet from the IP DSCP of the received unlabeled packet, based on the DSCP classifier policy applied to the ingress subinterface (or the default DSCP classifier policy, if there is no explicit association). If there is an mpls-tc classifier policy applied to the ingress subinterface, it has no effect.
- If there is a DSCP rewrite policy applied to the egress subinterface, it causes the IP header DSCP value to be rewritten before the egress MPLS encapsulation is applied.
- If there is no mpls-tc rewrite policy associated with the egress subinterface, EXP=0 is written into all pushed labels.
- If there is an mpls-tc rewrite policy associated with the egress subinterface, and it matches the forwarding-class (and possibly also the drop-probability) of the packet, the EXP provided by the mapping rule is written into the EXP field of all pushed labels.
- If ECN is enabled globally, and the packet hits an ECN slope in a congested queue such that the ECN marking should be '11', the ECN field of the IP packet is modified accordingly, and the DSCP field is also remarked according to the ecn-dscp-policy.

3.6.2 Transit LSR

When an SR Linux router, acting as transit LSR, matches an MPLS packet to a swap ILM entry:

- The transit LSR determines the forwarding-class and drop-probability of the packet from the EXP in the topmost label stack entry of the received labeled packet (before popping), based on the mpls-tc classifier policy applied to the ingress subinterface (or the default mpls-tc classifier policy, if there is no explicit association).
- If there is a DSCP classifier policy applied to the ingress subinterface, it has no effect on the classification of the packet.
- If there is a DSCP rewrite policy applied to the egress subinterface, it has no effect on the transmitted MPLS packet.
- If there is no mpls-tc rewrite policy associated with the egress subinterface, the classified FC of the packet is written as a value 0-7 into the EXP field of all pushed labels. This does not guarantee that the EXP of the popped labels matches the EXP of the pushed labels (that is, if a non-default mpls-tc classifier policy is applied to the ingress subinterface).
- If there is an mpls-tc rewrite policy associated with the egress subinterface, and it matches the forwarding-class (and possibly also the drop-probability) of the packet, the EXP provided by the mapping rule is written into the EXP field of all pushed labels.
- If ECN is enabled globally, it has no effect on the MPLS packet. The MPLS packet is considered non-ECT capable even if the buried IP ECN bits indicate otherwise. The IP ECN field is not modified.

3.6.3 PHP LSR

When an SR Linux router, acting as PHP LSR, matches an MPLS packet to a pop and swap-to-implicit-null ILM entry:

- The PHP LSR determines the forwarding-class and drop-probability of the packet from the EXP in the topmost label stack entry of the received labeled packet (before popping), based on the mpls-tc classifier policy applied to the ingress subinterface (or the default mpls-tc classifier policy, if there is no explicit association). If there is a DSCP classifier policy applied to the ingress subinterface, it has no effect on the classification of the packet.
- If there is an mpls-tc rewrite policy applied to the egress subinterface, it has no effect on the transmitted IP packet.

- If there is no DSCP rewrite policy associated with the egress subinterface, the DSCP field of the IP payload packet is transmitted unchanged. There is no attempt to copy the EXP field into the IP DSCP of the IP payload packet.
- If there is a DSCP rewrite policy associated with the egress subinterface, and it matches the forwarding-class (and possibly also the drop-probability) of the packet, the DSCP provided by the mapping rule is written (as an override) into the DSCP field in the transmitted IP packet. This is consistent with the uniform model of RFC 3270.
- If ECN is enabled globally, it has no effect on the PHP packet. The PHP packet is considered non-ECT capable even if the IP ECN bits indicate otherwise. The IP ECN field is not modified.

3.6.4 Egress LER

When an SR Linux router, acting as egress LER, matches an MPLS packet to a pop ILM entry that leads to all labels being popped:

- The egress LER determines the forwarding-class and drop-probability of the packet from the EXP in the topmost label stack entry of the received labeled packet (before popping), based on the mpls-tc classifier policy applied to the ingress subinterface (or the default mpls-tc classifier policy, if there is no explicit association).
If there is a DSCP classifier policy applied to the ingress subinterface, it has no effect on the classification of the packet.
- If there is an mpls-tc rewrite policy applied to the egress subinterface, it has no effect on the transmitted IP packet.
- If there is no DSCP rewrite policy associated with the egress subinterface, the DSCP field of the IP payload packet is transmitted unchanged. There is no attempt to copy the EXP field into the IP DSCP of the IP payload packet. This is consistent with the pipe model of RFC 3270.
- If there is a DSCP rewrite policy associated with the egress subinterface, and it matches the forwarding-class (and possibly also the drop-probability) of the packet, the DSCP provided by the mapping rule is copied into the IP DSCP of the transmitted IP packet, overwriting the previous value. This is consistent with the uniform model of RFC 3270.
- If ECN is enabled globally, it has no effect on the terminating MPLS packet. The terminating packet is considered non-ECT capable even if the IP ECN bits indicate otherwise. The IP ECN field is not modified.

Note that the DSCP marking of terminating MPLS traffic cannot be decoupled from the DSCP marking of transit IP traffic through the same egress subinterface.

3.6.5 Default MPLS Traffic-Class Classifier Policy

Table 3 shows the default mpls-tc classifier policy.

Table 3 Default mpls-tc classifier policy

Traffic class (EXP)	Forwarding class	Drop probability
0	0	low
1	1	low
2	2	low
3	3	low
4	4	low
5	5	low
6	6	low
7	7	low

3.7 Show reports for MPLS tunnel tables

You can issue a **show** command to display information about MPLS tunnel table entries. You can adjust the output of the report to filter by address type, encapsulation type, tunnel type, and destination prefix.

The following is an example of output from the **show** report:

```
--{ * candidate shared default }--[ ]--
# show network-instance default tunnel-table all
=====
IPv4 tunnel table of network-instance "default"
+-----+-----+-----+-----+-----+-----+-----+-----+
| IPv4 Prefix | Encap type | Tunnel type | FIB | ID | Metric | Pref | Next-hop (Type) |
+-----+-----+-----+-----+-----+-----+-----+
| 5.5.5.5/32 | vxlan      | vxlan      | Y   | 65538 | 1000    | 10   | 2.2.2.2 (tunnel) |
| 5.5.5.5/32 | mpls      | ldp        | Y   | 0     | 10      | 9    | 10.0.0.3 (mpls) |
|           |           |           | Y   |       |         |      | 10.0.0.4 (mpls) |
|           |           |           | Y   |       |         |      | 10.0.0.5 (mpls) |
| 6.6.6.6/32 | mpls      | ldp        | Y   | 0     | 10      | 9    | 10.0.0.3 (mpls) |
+-----+-----+-----+-----+-----+-----+-----+
1 VXLAN tunnels, 1 active, 0 inactive
2 LDP tunnels, 2 active, 0 inactive
3 Total Tunnels, 3 active, 0 inactive
```

=====

4 Configuring LDP

This chapter provides information about configuring Label Distribution Protocol (LDP) on SR Linux. It contains the following topics:

- [Enabling LDP](#)
- [Configuring LDP neighbor discovery](#)
- [Configuring LDP peers](#)
- [Configuring a label block for LDP](#)
- [Configuring longest-prefix match for IPv4 FEC resolution](#)
- [Configuring load balancing over equal-cost paths](#)
- [Configuring graceful restart helper capability](#)
- [Configuring LDP-IGP synchronization](#)

4.1 Enabling LDP

LDP must be enabled for the protocol to be active.

Example:

The following example administratively enables LDP for the default network-instance:

```
--{ * candidate shared default }--[ ]--
# info network-instance default protocols ldp admin-state
  network-instance default {
    protocols {
      ldp {
        admin-state enable
      }
    }
  }
```

4.2 Configuring LDP neighbor discovery

LDP neighbor discovery allows SR Linux to discover and connect to LDP peers without manually specifying the peers. SR Linux supports basic LDP discovery for discovering LDP peers, using multicast UDP hello messages.

Example:

The following example configures LDP neighbor discovery for a network-instance and enables it for a subinterface. The **hello-interval** parameter specifies the number of seconds between LDP link hello messages. The **hello-holdtime** parameter specifies how long the LDP link hello adjacency is maintained in the absence of link hello messages from the LDP neighbor.

```
--{ * candidate shared default }--[ ]--
# info network-instance default protocols ldp discovery
  network-instance default {
    protocols {
      ldp {
        discovery {
          interfaces {
            hello-holdtime 30
            hello-interval 10
            interface ethernet-1/1.1 {
              ipv4 {
                admin-state enable
              }
            }
          }
        }
      }
    }
  }
}
```

4.3 Configuring LDP peers

You can configure settings that apply to connections between the SR Linux and LDP peers, including session keepalive parameters. For individual LDP peers, you can configure the maximum number of FEC-label bindings that can be accepted by the peer.

If LDP receives a FEC-label binding from a peer that puts the number of received FECs from this peer at the configured FEC limit, the peer is put into overload. If the peer advertised the Nokia-overload capability (if it is another SR Linux router or an SR OS device) then the overload TLV is transmitted to the peer and the peer stops sending any further FEC-label bindings. If the peer did not advertise the Nokia-overload capability, then no overload TLV is sent to the peer. In either case, the received FEC-label binding is deleted.

Example:

The following example configures settings for LDP peers. The **session-keepalive-holdtime** parameter specifies the number of seconds an LDP session can remain inactive (no LDP packets received from the peer) before the LDP session is terminated and the corresponding TCP session closed. The **session-keepalive-interval** parameter specifies the number of seconds between LDP keepalive messages. SR Linux sends LDP keepalive messages at this interval only when no other LDP packets are transmitted over the LDP session.

For an individual LDP peer, indicated by its LSR ID and label space ID, a FEC limit is specified. SR Linux deletes FEC-label bindings received from this peer beyond this limit.

```
--{ * candidate shared default }--[ ]--
# info network-instance default protocols ldp peers
  network-instance default {
    protocols {
      ldp {
        peers {
          session-keepalive-holdtime 240
          session-keepalive-interval 90
        }
        peer 1.1.1.1 label-space-id 1254 {
          fec-limit 1024
        }
      }
    }
  }
}
```

4.4 Configuring a label block for LDP

LDP must be configured with a reference to a predefined range of labels, called a label block. A label block configuration includes a start-label value and an end-label value. LDP uses labels in the range between the start-label and end-label in the label block.

A label block can be static or dynamic. See [Static and dynamic label blocks](#) for information about each type of label block and how to configure them. LDP requires a *dynamic, non-shared* label block.

Example:

The following example configures LDP to use a dynamic label block named `d1`. The dynamic label block is configured with a start-label value and an end-label value. See [Configuring label blocks](#) for an example of a dynamic label block configuration.

```
--{ * candidate shared default }--[ ]--
# info network-instance default protocols ldp dynamic-label-block
```

```

network-instance default {
  protocols {
    ldp {
      dynamic-label-block dl
    }
  }
}

```

4.5 Configuring longest-prefix match for IPv4 FEC resolution

By default, SR Linux supports /32 IPv4 FEC resolution using IGP routes. You can optionally enable longest-prefix match for IPv4 FEC resolution. When this is enabled, IPv4 prefix FECs can be resolved by less-specific IPv4 routes in the route table, as long as the prefix bits of the route match the prefix bits of the FEC. The IP route with the longest prefix match is the route that is used to resolve the FEC.

Example:

The following example enables longest-prefix match for IPv4 FEC resolution:

```

--{ * candidate shared default }--[ ]--
# info network-instance default protocols ldp ipv4
network-instance default {
  protocols {
    ldp {
      ipv4 {
        fec-resolution {
          longest-prefix true
        }
      }
    }
  }
}

```

4.6 Configuring load balancing over equal-cost paths

ECMP support for LDP on SR Linux performs load balancing for LDP-based LSPs by using multiple outgoing next-hops for an IP prefix on ingress and transit LSRs. You can specify the maximum number of next-hops (up to 64) to be used for load balancing toward a specific FEC.

Example:

The following example configures the maximum number of next-hops the SR Linux can use for load balancing toward a FEC:

```
--{ * candidate shared default }--[ ]--
# info network-instance default protocols ldp multipath
  network-instance default {
    protocols {
      ldp {
        multipath {
          max-paths 64
        }
      }
    }
  }
```

4.7 Configuring graceful restart helper capability

Graceful restart allows a router that has restarted its control plane but maintained its forwarding state to restore LDP with minimal disruption.

To do this, the router relies on LDP peers, which have also been configured for graceful restart, to maintain forwarding state while the router restarts. LDP peers configured in this way are known as graceful restart helpers.

You can configure the SR Linux to operate as a graceful restart helper for LDP. When the graceful restart helper capability is enabled, the SR Linux advertises to its LDP peers by carrying the fault tolerant (FT) session TLV in the LDP initialization message, which assists the LDP peer to preserve its LDP forwarding state across the restart.

Example:

The following example enables the graceful restart helper capability for LDP. The **max-reconnect-time** parameter specifies the number of seconds the SR Linux waits for the remote LDP peer to reconnect after an LDP communication failure. The **max-recovery-time** parameter specifies the number of seconds the SR Linux router preserves its MPLS forwarding state after receiving the LDP initialization message from the restarted LDP peer.

```
--{ * candidate shared default }--[ ]--
# info network-instance default protocols ldp graceful--restart
  network-instance default {
    protocols {
      ldp {
        graceful-restart {
          helper-enable true
          max-reconnect-time 180
          max-recovery-time 240
        }
      }
    }
  }
```

```

    }
  }
}

```

4.8 Configuring LDP-IGP synchronization

You can configure synchronization between LDP and interior gateway protocols (IGPs). LDP-IGP synchronization is supported for IS-IS and OSPF.

When LDP-IGP synchronization is configured, LDP notifies the IGP to advertise the maximum cost for a link whenever the LDP hello adjacency goes down, the LDP session goes down, or LDP is not configured on an interface.

When LDP-IGP synchronization is configured:

- If a session goes down, the IGP increases the metric of the corresponding interface to max-metric.
- When the LDP adjacency is reestablished, the IGP starts a hold-down timer (default 60 seconds). When this timer expires, the IGP restores the normal metric, if it has not been restored already.
- When LDP informs the IGP that all label-FEC mappings have been received from the peer, the IGP can be configured to immediately restore the normal metric, even if time remains on the hold-down timer.

LDP-IGP synchronization does not take place on LAN interfaces unless the IGP has a point-to-point connection over the LAN, and does not take place on IGP passive interfaces.

Example:

The following example configures LDP-IGP synchronization for an IS-IS instance. If the LDP adjacency goes down, the IGP increases the metric of the corresponding interface to max-metric. If the adjacency is subsequently reestablished, the IGP waits for the amount of time configured with the **hold-down-timer** parameter before restoring the normal metric.

When the **end-of-lib** parameter is set to true, it causes the IGP to restore the normal metric when all label-FEC mappings have been received from the peer, even if time remains in the hold-down timer.

```

--{ * candidate shared default }--[ ]--
# info network-instance default protocols isis instance i1 ldp-synchronization
  network-instance default {
    protocols {

```

```
isis {  
    instance i1 {  
        ldp-synchronization {  
            hold-down-timer 120  
            end-of-lib true  
        }  
    }  
}
```


Customer Document and Product Support



Customer Documentation

[Customer Documentation Welcome Page](#)



Technical Support

[Product Support Portal](#)



Documentation Feedback

[Customer Documentation Feedback](#)

