Nokia Service Router Linux

Release 24.3

Troubleshooting Toolkit

3HE 20236 AAAA TQZZA
Edition: 01
March 2024

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.

Table of contents

1 About this guide

This document describes how to use and configure diagnostic tools for the Nokia Service Router Linux (SR Linux).

This document is intended for network technicians, administrators, operators, service providers, and others who need to understand how to use and configure diagnostic tools.

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

Configuration and command outputs shown in this guide are examples only; actual displays may differ depending on supported functionality and user configuration.

1.1 Precautionary and information messages

The following are information symbols used in the documentation.

DANGER: Danger warns that the described activity or situation may result in serious personal injury or death. An electric shock hazard could exist. Before you begin work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.

WARNING: Warning indicates that the described activity or situation may, or will, cause equipment damage, serious performance problems, or loss of data.

Caution: Caution indicates that the described activity or situation may reduce your component or system performance.

Note: Note provides additional operational information.

Tip: Tip provides suggestions for use or best practices.

1.2 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**

- 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 What’s new

Topic	Location
ACL packet capture filter configuration now aligns with OpenConfig model.	Interactive traffic-monitoring tool Configuring mirroring sources
Mirroring enhancements for 7250 IXR 6e/10e platforms	Mirror sources Mirror destinations
On 7250 IXR platforms, one IPv6 sFlow collector is now supported.	sFlow collector reporting Configuring sFlow collectors Sampled data and counter examples
On 7250 IXR platforms, sFlow frame sample sizes of 256 or 512 bytes are supported.	sFlow sampling Configuring the sFlow agent

3 sFlow

sFlow is used to monitor data traffic flows traversing different points in a network. The sFlow functionality uses an sFlow agent and an sFlow collector. The agent is software that runs on a network element and samples and reports flow headers and statistics. The collector is software that typically runs on a remote server and receives the flow headers and statistics from one or more sFlow agents.

Sampling and reporting are accomplished as the sFlow agent running on a network element takes periodic samples of ingress traffic and reports the data to one or more collectors. The network element does not need to maintain a local flow cache. Instead, the sampled header information is immediately sent to the collector without additional processing.

SR Linux supports sFlow version 5 behavior and formats. On 7250 IXR chassis-based systems, sFlow is implemented in hardware. On 7220 IXR systems, sFlow functionality is implemented in software. sFlow behavior is identical on both platforms, with the following exceptions:

- DSCP configuration is supported only on 7250 IXR systems.
- Frame sample sizes of 256 or 512 bytes are supported only on 7250 IXR systems.
- IPv6 sFlow collector configuration is supported only on 7250 IXR systems.

3.1 sFlow sampling

sFlow works by sampling flow data and reporting the samples to the configured sFlow collectors. Based on the configured system sampling rate, the forwarding plane samples ingress packet flows and sends the sampled headers to the sFlow agent in the control plane.

All ingress packets are subject to sampling. By default, 256 bytes are sampled from each packet. Each sample includes the following:

- 7220 IXR systems – samples include the top 256 bytes of the sampled packet, starting at the outer Ethernet header
- 7250 IXR systems – samples include the top 256 or 512 bytes of the sampled packet, starting at the outer Ethernet header

The sampled packets are sent to the configured sFlow collectors with the sampled data in sFlow raw packet data format.

For sampled IPv4 packets, the IPv4 header data fields are sent with the raw data. For sampled IPv6 packets, the IPv6 header data fields are sent with the raw data.

3.2 sFlow collector reporting

sFlow reports sampled headers and statistics to the configured collectors using IP/UDP datagrams. UDP port 6343 is the default destination port, but you can optionally configure a different port. Sampled packets are sent as soon as the samples are taken, and interface statistics are sent at 10 second intervals. SR Linux supports up to eight remote IPv4 sFlow collectors or one remote IPv6 sFlow collector. IPv6 sFlow collector configuration is supported only on 7250 IXR systems. IPv4 and IPv6 sFlow collectors are mutually exclusive and cannot be configured simultaneously. Each collector can only have one IPv4 or IPv6 address. The flow and counter samples are aggregated in an sFlow datagram packet in software implementation.

sFlow DSCP settings

On 7220 IXR systems, flow and counter samples are assigned a non-configurable default DSCP value of 0.

On 7250 IXR systems:

- Flow samples are also assigned a default DSCP value of 0, but you can optionally assign a different DSCP value for flow samples that applies to all collectors.
- Counter samples are assigned a default DSCP value of 34, which cannot be modified.

3.3 sFlow counter samples

Another aspect of the sFlow agent is streaming of interface statistics to configured sFlow collectors. Statistics are only sent to a collector if sFlow has been enabled on an interface. Interface statistics are sent based on a default poll-interval of 10 seconds with a separate timer for each interface. When the interval expires. the current value of each associated statistics are sent to the configured collectors.

The interface counter sample contains:

- Interface index
- Interface type
- Interface speed
- Oper and admin status
- Input octets
- Input packets
- Input broadcast packets
- Input discards packets
- Output errors
- Output octets
- Output packets
- Output broadcast packets
- Output discards packets

3.4 Configuring the sFlow agent

Procedure

To configure the sFlow agent on the system, you enable sFlow, and optionally configure the sampling rate (by default, 1 out of every 10 000 packets) and sample size (by default, 256 bytes are sampled from each packet).

Example: Configuring the sFlow agent

The following example enables sFlow on the system and configures the system sampling rate and sample size. The polling interval is not configurable. The following sample size options apply:

- 7220 IXR-D2, D3, D4, D5, and 7220 IXR-H systems: 256 bytes
- 7250 IXR 6/10: 256 or 512 bytes

```
--{ * candidate shared default }--[  ]--
# info
    system {
        sflow {
            admin-state enable
            sample-rate 50000
            sample-size 512
        }
    }
```

3.5 Configuring sFlow collectors

Procedure

The sFlow agent sends sampled packets to sFlow collectors. You can configure up to eight IPv4 sFlow collectors or one IPv6 sFlow collector to receive the data. IPv6 sFlow collector configuration is supported only on 7250 IXR systems. IPv4 and IPv6 sFlow collectors are mutually exclusive and cannot be configured simultaneously. To configure an sFlow collector, you specify its IP address, associated network instance, and IP address to be used as the source IP address in sFlow packets sent from SR Linux to the collector. You can optionally specify a destination port (by default, this is UDP port 6343).

Note:
Configuring a network-instance is mandatory. Also, a collector cannot be reached using the **mgmt** network-instance.

Example: Configuring IPv4 sFlow collectors

The following example configures two IPv4 sFlow collectors. The IP address for each collector is configured, as well as its network instance and source IP address. Each collector receives all samples. The collector DSCP value for flow samples is also configured (applicable only on 7250 IXR systems). If no value is specified, the default DSCP value of 0 applies.

```
--{ * candidate shared }--[  ]--
#info system sflow
    system {
        sflow {
            dscp 14
            collector 1 {
                collector-address 10.50.4.1
                source-address 192.0.2.1
                network-instance default
            }
            collector 2 {
                collector-address 10.50.4.2
                source-address 10.1.5.2
                network-instance default
                port 4310
            }
```

```
        }
    }
```

Example: Configuring an IPv6 sFlow collector

The following example configures one IPv6 sFlow collector. The IP address for the collector is configured, as well as its network instance and source IP address. The collector receives all samples.

Note: Only one IPv6 collector with a **collector-id** value of 1 can be configured.

```
--{ * candidate shared default }--[  ]--
# info system sflow
    system {
        sflow {
            collector 1 {
                collector-address 2001:db8::1
                network-instance default
                source-address 2001:db8::2
            }
        }
    }
```

3.6 Configuring sFlow for an interface

Procedure

When sFlow is configured for an interface, the ingress packets are taken for sampling according to the **sample-rate**.

Example: Configuring sFlow for an interface

The following example enables sFlow on an interface.

```
--{ * candidate shared }--[  ]--
    interface ethernet-1/1 {
        sflow {
            admin-state enable
        }
    }
```

3.7 Displaying the state of the sFlow agent

Procedure

To display the system-wide state of the sFlow agent, including any sFlow parameters, collector configuration, and general statistics, use the **info from state** command in candidate or running mode, or the **info** command in state mode.

Example: Info from state command

```
# info from state system sflow
    system {
```

```
        sflow {
            admin-state enable
            sample-rate 1000
            sample-size 256
            collector 1 {
                collector-address 10.1.1.24
                network-instance default
                source-address 10.0.0.1
                port 6343
                next-hop 172.24.71.65
            }
            statistics {
                total-samples-taken 5457
                total-sent-packets 26800
            }
        }
    }
```

3.8 Displaying the status of the sFlow agent

Procedure

Use the **show system sflow status** command in show mode to display the general status of the sFlow agent:

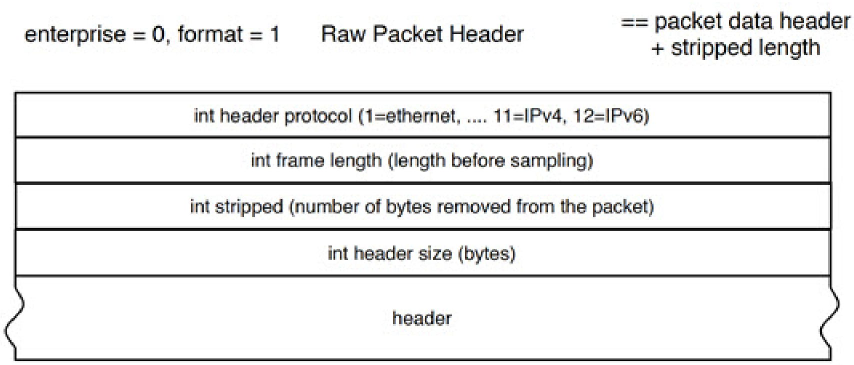
Example: Show system sflow status command

```
--{ running }--[  ]--
# enter show
# show system sflow status
-------------------------------------
Admin State             : enable
Sample Rate             : 10000
Sample Size             : 256
DSCP                    : 0
Total Samples           : 0
Total Collector Packets: 3269158
-------------------------------------
collector-id      : 8
collector-address: 172.10.10.10
network-instance : default
source-address    : 10.0.0.1
port              : 6343
next-hop          : 172.24.71.65
-------------------------------------
```

3.9 sFlow formats

Figure 1: Raw packet header shows an example of a raw packet header for an sFlow format.

Figure 1: Raw packet header



3.10 Sampled data and counter examples

The following is an example of IPv4 flow sample data:

Example: IPv4 flow sample data

```
InMon sFlow
    Datagram version: 5
    Agent address type: IPv4 (1)
    Agent address: 10.0.0.1
    Sub-agent ID: 2
    Sequence number: 0
    SysUptime: 0
    NumSamples: 1
    Flow sample, seq 0
        0000 0000 0000 0000 0000 .... .... .... = Enterprise: standard sFlow (0)
        .... .... .... .... .... 0000 0000 0001 = sFlow sample type: Flow sample (1)
        Sample length (byte): 141
        Sequence number: 0
        0000 0000 .... .... .... .... .... .... = Source ID class: 0
        .... .... 0000 0000 0000 0000 0011 0110 = Index: 54
        Sampling rate: 1 out of 5 packets
        Sample pool: 0 total packets
        Dropped packets: 0
        Input interface (ifIndex): 54
        .000 0000 0000 0000 0000 0000 0011 0110 = Output interface (ifIndex): 54
        Flow record: 1
        Raw packet header
            0000 0000 0000 0000 0000 .... .... .... = Enterprise: standard sFlow (0)
            Format: Raw packet header (1)
            Flow data length (byte): 101
            Header protocol: Ethernet (1)
            Frame Length: 98
            Payload removed: 0
            Original packet length: 85
            Header of sampled packet:
                000c00020000000000111111080045000052000000004006...
                Ethernet II, Src: 00:00:00_11:11:11 (00:00:00:11:11:11),
                    Dst: BebIndus_02:00:00 (00:0c:00:02:00:00)
```

```
                    Destination: BebIndus_02:00:00 (00:0c:00:02:00:00)
                    Source: 00:00:00_11:11:11 (00:00:00:11:11:11)
                    Type: IPv4 (0x0800)
                Internet Protocol Version 4, Src: 10.100.1.2, Dst: 10.1.1.2
                    0100 .... = Version: 4
                    .... 0101 = Header Length: 20 bytes (5)
                    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
                    Total Length: 82
                    Identification: 0x0000 (0)
                    Flags: 0x00
                    Fragment offset: 0
                    Time to live: 64
                    Protocol: TCP (6)
                    Header checksum: 0x35a1 [validation disabled]
                    [Header checksum status: Unverified]
                    Source: 10.100.1.2
                    Destination: 10.1.1.254
                    [Source GeoIP: Unknown]
                    [Destination GeoIP: Unknown]
                Transmission Control Protocol, Src Port: 0, Dst Port: 0, Seq: 0
                LBT-TCP Protocol
                LBMC Protocol
                [Unreassembled Packet: LBT-TCP]
```

The following is an example of IPv6 flow sample data:

Example: IPv6 flow sample data

```
InMon sFlow
    Datagram version: 5
    Agent address: 3000::2 (3000::2)
    Sub-agent ID: 24
    Sequence number: 1011
    SysUptime: 63684188
    NumSamples: 1
    Flow sample, seq 2368
        Enterprise: standard sFlow (0)
        sFlow sample type: Flow sample (1)
        Sample length (byte): 568
        Sequence number: 2368
        Source ID class: 0 index: 704510
        Sampling rate: 1 out of 1 packets
        Sample pool: 0 total packets
        Dropped packets: 0
        Input interface: ifIndex 134922238
        Output interface: ifIndex 0
        Flow record: 1
        Raw packet header
            Enterprise: standard sFlow (0)
            Format: Raw packet header (1)
            Flow data length (byte): 528
            Header protocol: Ethernet (1)
            Frame Length: 125 bytes
            Payload removed: 0 bytes
            Header of sampled packet: 01005e000002000103ff02018100c064080045c0006b3005...
                Ethernet II, Src: 3com_ff:02:01 (00:01:03:ff:02:01), Dst:
                IPv4mcast_00:00:02 (01:00:5e:00:00:02)
                802.1Q Virtual LAN, PRI: 6, CFI: 0, ID: 100
                Internet Protocol Version 4, Src: 192.35.1.1 (192.35.1.1),
                Dst: 224.0.0.2 (224.0.0.2)
                User Datagram Protocol, Src Port: ldp (646), Dst Port: ldp (646)
                Label Distribution Protocol
                    Version: 1
```

```
                    PDU Length: 75
                    LSR ID: 3.3.3.1 (3.3.3.1)
                    Label Space ID: 0
                    Hello Message
```

The following is a counter sample example:

Example: Counters sample

```
InMon sFlow
    Datagram version: 5
    Agent address: 10.0.0.1 (10.0.0.1)
    Sub-agent ID: 0
    Sequence number: 8
    SysUptime: 6548000
    NumSamples: 1
    Counters sample, seq 1
        Enterprise: standard sFlow (0)
        sFlow sample type: Counters sample (2)
        Sample length (byte): 108
        Sequence number: 1
        Source ID type: 64
        Source ID index: 49150
        Counters records: 1
        Generic interface counters
            Enterprise: standard sFlow (0)
            Format: Generic interface counters (1)
            Flow data length (byte): 88
            Interface index: 1073790974
            Interface Type: 6
            Interface Speed: 25600
            IfDirection: Full-Duplex
            IfAdminStatus: Up
            IfOperStatus: Up
            Input Octets: 0
            Input Packets: 0
            Input Multicast Packets: 0
            Input Broadcast Packets: 0
            Input Discarded Packets: 0
            Input Errors: 0
            Input Unknown Protocol
        Packets: 0
            Output Octets: 0
            Output Packets: 0
            Output Multicast Packets: 0
            Output Broadcast Packets: 0
            Output Discarded Packets: 0
            Output Errors: 0
            Promiscuous Mode: 0
```

4 Interactive traffic-monitoring tool

SR Linux features an interactive traffic-monitoring tool that allows you to capture and monitor traffic based on 5-tuple match criteria. The match criteria is injected into a packet capture filter ACL entry that is applied to all subinterfaces; information from matching packets can be displayed on screen or directed to a file.

4.1 Using the interactive traffic-monitoring tool

Procedure

You can specify the match criteria either by using the **tools system traffic-monitor** CLI command, or by defining capture-filter ACL entries.

If you use the **tools system traffic-monitor** command to specify the match criteria, SR Linux dynamically creates a capture-filter entry with the match criteria. Packets that match the capture-filter entry are sent to the traffic-monitoring tool running on the CPM and displayed until the traffic-monitoring tool is exited, at which time the dynamically created capture-filter entries are removed.

Use the following syntax to configure the **tools system traffic-monitor** command:

tools system traffic-monitor [source-address <*ip-addr/len*>**] [destination-address** <*ip-addr/len*>**] [protocol** <*proto-val*>**] [source-port** <*value* | *range*>**] [destination-port** <*value* | *range*>**] [verbose] [output-file** <*file-name*>**] [hex-output]**

The command parameters are described in Table 1: Traffic monitoring command parameters.

Table 1: Traffic monitoring command parameters

Command/parameter	Description
tools system traffic-monitor	Initiates an interactive monitor session
source-address <*ip-addr/len*>	Source IP address (IPv4 or IPv6) prefix and netmask length value. For example: 10.10.11.0/24
destination-address <*ip-addr/len*>	Destination IP address (IPv4 or IPv6) prefix and netmask length value. For example: 10.10.20.0/24
protocol <*proto-val*>	Specifies the protocol type value to match (required if either port values are specified)
source-port <*value* | *range*>	Source port integer value or port range in the format of port1..port2
destination-port <*value* | *range*>	Destination port integer value or port range in the format of port1..port2

Command/parameter	Description
verbose	Displays detailed output
output-file *<file-name>*	Directs output to a file
hex-output	Displays output in hex format

If you specify the match criteria by defining capture-filter ACL entries, starting the traffic-monitoring tool with the **tools system traffic-monitor** command causes the system to send packets that match the defined capture-filter entries to the CPM and display them until the traffic-monitoring tool is exited. Unlike the dynamically created capture-filter entries, the defined capture-filter entries are not removed from the system when the traffic-monitoring tool is exited.

There is one packet capture filter for IPv4 traffic and another packet capture filter for IPv6 traffic. The default IPv4 packet capture filter copies no IPv4 packets, and the default IPv6 packet capture filter copies no IPv6 packets. To configure a packet capture filter, you create an IPv4 or IPv6 ACL filter named `capture` and specify match conditions and actions.

Note: The name **capture** is reserved for packet capture filters; an ACL named **capture** cannot be associated with any interface.

The following is an example of a capture-filter ACL entry:

Example: Capture filter ACL entry

```
--{ * candidate shared default }--[  ]--
# info acl acl-filter capture type ipv4
    acl {
        acl-filter capture type ipv4 {
            entry 1 {
                match {
                    ipv4 {
                        protocol icmp
                        destination-ip {
                            address 10.1.1.1
                            mask 255.255.255.255
                        }
                        source-ip {
                            address 10.2.2.2
                            mask 255.255.255.255
                        }
                    }
                }
                action {
                    copy {
                    }
                }
            }
        }
    }
```

Capture filters are applied to traffic after any subinterface filters, but before CPM filters. If a packet is dropped by a subinterface filter, it is not evaluated by a capture filter.

Only a single instance of the traffic-monitoring tool can be running at a time.

If no capture-filter entries are already defined, you must specify the match criteria with the **tools system traffic-monitor** command. If capture-filter entries are already defined, match criteria specified with the **tools system traffic-monitor** command is ignored.

4.1.1 Monitoring ICMP Packets

Procedure

The following is an example of using the traffic-monitoring tool to monitor ICMP packets. In this example, information about ICMP packets with source address 10.1.1.1/32 and destination address 10.2.2.2/32 is displayed in the monitor window, including the arrival time and source port (ethernet-1/20.1) of each packet. The traffic-monitoring tool captures ICMP packets until you press Ctrl-C.

Example: Traffic-monitoring tool

```
# tools system traffic-monitor destination-address 10.1.1.1/32 source-address 10.2.2.2/32 protocol
 icmp
Capturing on 'monit'
1 0.000   ethernet-1/20.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
2 1.803   ethernet-1/20.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
3 2.895   ethernet-1/20.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
4 3.749   ethernet-1/20.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
5 4.250   ethernet-1/20.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
6 5.759   ethernet-1/20.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
7 6.644   ethernet-1/20.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
^C
7 packets captured
Command execution aborted : 'tools system traffic-monitor destination-address 10.1.1.1/32 source-
address 10.2.2.2/32 protocol icmp'
```

When you execute the **tools system traffic-monitor** command in the example above, it dynamically creates the following traffic monitoring policy:

```
    acl {
        acl-filter capture type ipv4 {
            entry 1 {
                match {
                    ipv4 {
                        protocol icmp
                        destination-ip {
                            address 10.1.1.1
                            mask 255.255.255.255
                        }
                        source-ip {
                            address 10.2.2.2
                            mask 255.255.255.255
                        }
                    }
                }
                action {
                    copy {
                    }
                }
            }
        }
    }
```

When you terminate the command by pressing Ctrl-C, the dynamically created traffic monitoring policy is removed from all ingress interfaces.

4.1.2 Displaying verbose output

Procedure

If you include the **verbose** option in the **tools system traffic-monitor** command, it displays the header fields and additional information from the shim header, followed by the original packet.

Example: Verbose output

The following example shows verbose output for an ICMP packet:

```
# tools system traffic-monitor destination-address 10.1.1.1/32 source-address 10.2.2.2/32 protocol
 icmp verbose
Frame 1: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
    Interface id: 0 (monit)
        Interface name: monit
    Encapsulation type: Ethernet (1)
    Arrival Time: Jan  4, 2098 19:53:01.144789891 UTC
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: -255263715.144789891 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 146 bytes (1168 bits)
    Capture Length: 146 bytes (1168 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:srlinux:eth:ethertype:ip:icmp:data]
Srlinux Packet
    Ingress Port: ethernet-1/20.1
    Padding: 000000
Ethernet II, Src: 00:00:5e:00:53:d2, Dst: 00:00:5e:00:53:41
    Destination: 00:00:5e:00:53:41
        Address: 00:00:5e:00:53:41
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: 00:00:5e:00:53:d2
        Address: 00:00:5e:00:53:d2
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.20.20.20, Dst: 10.10.10.10
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 84
    Identification: 0xa166 (41318)
    Flags: 0x0000
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..0. .... .... .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 63
    Protocol: ICMP (1)
    Header checksum: 0x9e07 [validation disabled]
```

```
    [Header checksum status: Unverified]
    Source: 10.2.2.2
    Destination: 10.1.1.1
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xd01f [correct]
    [Checksum Status: Good]
    Identifier (BE): 10408 (0x28a8)
    Identifier (LE): 43048 (0xa828)
    Sequence number (BE): 1352 (0x0548)
    Sequence number (LE): 18437 (0x4805)
    Timestamp from icmp data: Jan  4, 2098 19:53:01.000000000 UTC
    [Timestamp from icmp data (relative): 0.144789891 seconds]
    Data (48 bytes)
0000  5a 30 02 00 00 00 00 00 10 11 12 13 14 15 16 17   Z0..............
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
    Data: 5a30020000000000101112131415161718191a1b1c1d1e1f…
    [Length: 48]
```

4.1.3 Capturing packets to a file

Procedure

You can direct the captured packets to a file, which can be used as a source for the SR Linux packet trace utility or for Wireshark.

Example: Directing captured packets

The following example directs information about ICMP packets with source address 10.1.1.1/32 and destination address 10.2.2.2/32 to a `.pcap` file.

```
# tools system traffic-monitor destination-address 10.1.1.1/32 source-address 10.2.2.2/32
 protocol icmp output-file /home/linuxadmin/ICMP.pcap
Capturing on 'monit'
6 packets captured
Command execution aborted : 'tools system traffic-monitor destination-address 10.1.1.1/32
 source-address 10.2.2.2/32 protocol icmp output-file /home/linuxadmin/ICMP.pcap '
```

Before opening the `.pcap` file, remove the shim header (the first 48 bytes of the file). For example:

```
$ editcap -C 0:48  /home/linuxadmin/ICMP.pcap /home/linuxadmin/ICMP_chopped.pcap
```

4.1.4 Capturing bidirectional transit traffic

Procedure

The 5-tuple matching criteria defined in a **tools system traffic-monitor** command applies in one direction only. To capture traffic in both directions, you define capture filters for each direction, then start the traffic-monitoring tool, which applies both capture filters on all ports.

Example: Capturing bidirectional transit traffic

The following example defines two capture filter entries: one that matches traffic with source address 10.1.1.1/32 and one that matches traffic with destination address 10.2.2.2/32.

```
--{ * candidate shared default }--[  ]--
# info acl acl-filter capture type ipv4
    acl {
        acl-filter capture type ipv4 {
            entry 10 {
                match {
                    ipv4 {
                        source-ip {
                            address 10.1.1.1
                            mask 255.255.255.255
                        }
                    }
                }
                action {
                    copy {
                    }
                }
            }
            entry 20 {
                match {
                    ipv4 {
                        destination-ip {
                            address 10.2.2.2
                            mask 255.255.255.255
                        }
                    }
                }
                action {
                    copy {
                    }
                }
            }
        }
    }
```

When you start the traffic-monitoring tool, it captures packets matching both filter entries. For example:

```
# tools system traffic-monitor
Capturing on 'monit'
1 0.000  ethernet-1/20.1 1.1.1.1 2.2.2.2  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
2 1.803  ethernet-1/21.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
3 2.895  ethernet-1/20.1 1.1.1.1 2.2.2.2  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
4 3.749  ethernet-1/21.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
5 4.250  ethernet-1/20.1 1.1.1.1 2.2.2.2  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
6 5.759  ethernet-1/21.1 2.2.2.2 1.1.1.1  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
7 6.644  ethernet-1/20.1 1.1.1.1 2.2.2.2  ICMP 146 Echo (ping) reply id=0x28a8, seq=119/30464, ttl=63
```

5 Switch fabric statistics

The switch fabric statistics tool allows you to monitor and troubleshoot common switch fabric issues at different points in the fabric.

The tool can be used to determine the current utilization level. Utilization data is displayed on a per-slot and line-card basis and includes aggregate line card/slot to switch fabric utilization (bidirectional).

See the *SR Linux Data Model Reference* for details on all switch fabric statistic related commands and descriptions of all parameters.

5.1 Displaying switch fabric statistics

Procedure

Use this procedure to display switch fabric statistics:

```
# enter show
```

```
# tools platform show-fabric-bandwidth
```

Example: Displaying switch fabric statistics

```
/platform/show-fabric-bandwidth:

    Slot     to-fabric Gbps   from-fabric Gbps
    ------------------------------------------
     1        2369             2370
     2        2393             2393
    ------------------------------------------
    Total     4762             4764
```

6 Packet-trace tool

The packet-trace tool is a troubleshooting command that allows the specification of a probe packet that is injected into the specified interface forwarding context. The tool records the forwarding destination or egress port for the probe packet, as well as any matched ACL records.

The packet-trace tool calculates the egress interfaces for an IP forward flow, while taking into account ECMP and LAG hashing.

The tool reports the following output:

- supplied input parameters
- calculated egress interface and port through which a packet with the specified fields is forwarded
- applied ACL (both ingress and egress)
- reason for a discarded packet

Note: Packet-trace is not supported on 7220 IXR-D4/D5 systems.

6.1 Configuring packet-trace tool commands

The **packet-trace** command is a tools command that reports the forwarding behavior for a test packet specified in one of the following formats:

- Scapy file format: file specifying the packet format in Scapy packet definition form
- base64 format: string specifying the packet to send in base64 format
- pcap file format: file containing pcap data

Only physical interface types can be used as the ingress interface for injected packets.

6.1.1 Configuring the packet-trace tool (using Scapy file format)

Procedure

Use this command to report the forwarding behavior for a specified test packet (file format) that contains a packet formatted in Scapy packet definition form:

\# **tools system packet-trace file** *<input file in Scapy format>* **interface** *<interface name>*

Packet trace command parameters for specifying an input file are described in Table 2: Packet trace command parameters using an input file.

Table 2: Packet trace command parameters using an input file

Command/parameter	Description
tools system packet-trace	Reports the forwarding behavior for a specified test packet (file format)
file *<file name>*	File containing the packet format in Scapy packet definition form. The format of the packet definition should match that of the Linux utility Scapy.
interface *<interface name>*	The name of the configured interface to inject the probe packet

Example: Scapy input file

```
# bash cat /tmp/p1.txt
Ether(dst="00:00:5E:00:53:D2",src="00:00:5E:00:53:41")/Dot1Q(vlan=100)/
IP(dst="10.1.5.1",src="192.0.2.1")/UDP(sport=6722,dport=6789)/"Hi"/
Raw(RandString(size=512))
```

Example: command

```
# tools system packet-trace file /tmp/p1.txt interface ethernet-1/1
```

Example: output (bridged)

```
Ether(dst="00:00:5E:00:53:D2",src="00:00:5E:00:53:41")/Dot1Q(vlan=100)/IP(dst=
"10.1.5.1",src="192.35.1.1")/UDP(sport=6722,dport=6789)/"Hi"/Raw(RandString(size=512))
Generated packet:
###[ Ethernet ]###
    dst       = 00:00:5e:00:53:d2
    src       = 00:00:5e:00:53:41
    type      = VLAN
###[ 802.1Q ]###
    prio      = 0
    id        = 0
    vlan      = 100
    type      = IPv4
###[ IP ]###
    version   = 4
    ihl       = 5
    tos       = 0x0
    len       = 542
    id        = 1
    flags     =
    frag      = 0
    ttl       = 64
    proto     = udp
    chksum    = 0x4ea8
    src       = 192.0.2.1
    dst       = 10.1.5.1
    \options   \
###[ UDP ]###
    sport     = 6722
    dport     = smc_https
    len       = 522
```

```
    chksum    = 0x251e
###[ Raw ]###
    load      = 'Hi9cmfMxg4lBV6iXRKbe3t2dUJyiGZb7s2GcTQ8YQ0A2PYnF8ntm45l
GqCezZ6ncYF4ijsc7hqjxSUjIJdq4YhhRrNSnyUsHkhehhSif
TpT1EEiQNOzNLWgF6DPdcQ078REyyjnI9hqzTNAk0XhgOmLtg55rkufD8nyOotgBgnz2mpQ0igLSEtYe84VDfdi
Cs5lWTvhGTYCClxsCYmXEozmSsWqBagdwHe1Ia0voCZ3deUUL6B7paAOb8ua5bZa44G7Z7LneJZ0YxH2Vjb
SqmeukaxyMrkg7NUIxs3aVIwD2jPqra3CBaxokvarX5TyIzNuK2qYeAwnjdzBZo2iZTonXom
JjoDWB2cqG61iEGPLNg5juC7PTa9fglirYgEI2T9rTm8gpTjG6ZgN90g3w0xOxBgwYsNfuXMqp7u9wR8fvfNa4Mm
ZseCC6UUKneSKK0zDyxyHgtSEKwHQgQA0HOh6wZttNQRFzST4YB0cFM1tTeo6mCgwAplYX8THGImjvis'
/system/packet-trace-base64:
================================================================================
Ingress information for Packet 3 Ingress Interface ethernet-1/1
================================================================================
Type                : Bridged
Interface           : ethernet-1/1 (4401020001)
Net Instance        : macvrf1
================================================================================
================================================================================
Egress information for Packet 3
================================================================================
Interface           : Flooded in macvrf1
Egress Net Instance : macvrf1
================= (routed)======================================================
```

Example: output (routed)

```
smac='00:00:5E:00:53:D2'
dmac= '00:00:5E:00:53:41'
Ether(src=smac,dst=dmac)/IP(src='192.0.2.1',dst='10.1.5.1')/ICMP()

Generated packet:
###[ Ethernet ]###
dst = 00:00:5e:00:53:41
src = 00:00:5e:00:53:d2
type = IPv4
###[ IP ]###
    version = 4
    ihl = 5
    tos = 0x0
    len = 28
    id = 1
    flags =
    frag = 0
    ttl = 64
    proto = icmp
    chksum = 0x7edc
    src = 192.0.2.1
    dst = 10.1.5.1
    \options \
###[ ICMP ]###
    type = echo-request
    code = 0
    chksum = 0xf7ff
    id = 0x0
    seq = 0x0
/system/packet-trace-base64:

================================================================================
Ingress information for Packet 1 Ingress Interface ethernet-4/29
================================================================================
Type                : Routed
Interface           : lag5 (14000000005)
Sub interface       : lag5.1
```

```
Net Instance         : red
Out Interface        : ethernet-4/22
Nexthop ip           : 192.0.52.1
================================================================================
================================================================================
Egress information for Packet 1 Egress Interface ethernet-4/22
================================================================================
Interface            : ethernet-4/22 (4404020016)
Sub interface        : ethernet-4/22.1
Mac Address          : 00:01:03:FF:00:08
================================================================================
```

6.1.2 Configuring the packet-trace tool (using base64 format)

Procedure

Use this command to report the forwarding behavior for a specified test packet using packets specified in base64 format:

\# **tools system packet-trace-base64 interface** *<interface name>* **packet** *<value>*

Packet trace command parameters for specifying base64 format are described in Table 3: Packet trace command parameters using base64 string format.

Table 3: Packet trace command parameters using base64 string format

Command/parameter	Description
tools system packet-trace-base64	Reports the forwarding behavior for a specified test packet (packet specified in base64 format)
interface *<interface name>*	The name of the configured interface to inject the probe packet
packet *<value>*	Packet format in base64 string format

Example: command (for routed)

```
# tools system packet-trace-base64 interface ethernet-1/1 packet
"RQAAOgABAABABnS4AQEBAQICAgIAFABQAAAAAAAAAABQAiAAqscAAEdFVCAvIEhUVFAvMS4wDQoNCg=="
```

Example: output (routed)

```
tools system packet-trace-base64 interface ethernet-1/3 packet
"MjBFMDlDNzlCQUUzMDAwMTA3RkYwMDAwMDgwMDQ1MDAwMDJFMDAwMDAwMDA0MDExNTA3REMwMz
kwMTA3NjQwMTA1MDExQTQyMUE4NTAwMUE1ODVGMDAwMTAyMDMwNDA1MDYwNzA4MDkwQTBCMEMwRDBFMEYxMD
ExM0VGMjA5OTM="
/system/packet-trace-base64:
================================================================================
Ingress information for Packet 77 Ingress Interface ethernet-1/2
================================================================================
Type          : Routed
Interface     : ethernet-1/2 (4401020002)
Sub interface : ethernet-1/2.1
Instance id   : 1
Out Interface : ethernet-1/1
```

```
Nexthop ip    : 10.1.5.1
================================================================================
================================================================================
Egress information for Packet 77 Egress Interface ethernet-1/1
================================================================================
Interface : ethernet-1/1 (4401020001)
Sub interface : ethernet-1/1.1
Mac Address : 00:00:5e:00:53:41
================================================================================
```

Example: command (for bridged)

```
tools system packet-trace-base64 packet UODvOurSAAED/wBBgQAAZAgARQACHgAB
AABAEU6owCMBAWQBBQEaQhqFAgpUvkhpQzVDODlhcUttaDVsRHNWRnk4WDRpYXRQbW81UllWenFweE9p
NEJNeXpPQWo5UktOVTRGNkFwTENhNVljNlFVMHViRTY2UUJzUkh5TWh0SHhQUTZOaFFTRk5LeXFKNGVn
VVZINDJlOFdBSmt1NlFZeHFicFRmZjEwdHVWdENwcENNMmQ5R1RCeHpseUY3aDZrQjBLMHRXNkF1a2Y0
QllNS3Jld2M5aUVGNGRUc1pPbEs0WVFEdkpxRjFOQ1BMMktXNjlnS212bXJmbTlZT2tHWE01MG9haTdp
R2l0amNzRHdkV3VBZEJ4OHJvek5tbnVQc2FCYVdPeVBWUjJBT0hVa1BrOW1jldwYTFDVXVOcU8xZzJk
RVExRXhBNFhaYUlnNlJLZjJvc2swMVJZektac0dKZEFUVnBaSkQzM2tnY2c4UDJnM0dYZFYzZnp4VTNH
bEtEQzhRUUlzQTJVYUJOODM4TWNiNmw3MUdZdGNuZlNDdGZFYlB0TU9OS2xxSejlhYWZHb3JaQzVMNFdw
TjZXRDVzZWlKelZtYWdrWUM2VThYY2dKWGpDSXJpR01lQjlobnY4RmFjNklDZnpROHFiZE5iZ1pTTTG9M
N0l0Tk4xZ1NmQ2JkeUE0RVFabHBGYlFEeVFFYUFJZUUycG9lbWRPU2x4a0FWYzBQU3kzZExEYWE=int
erface ethernet-1/1
```

Example: output (bridged)

```
A:rifa# tools system packet-trace-base64 packet UODvOurSAAED/wBBgQAAZAgARQACHgAB
AABAEU6owCMBAWQBBQEaQhqFAgpUvkhpQzVDODlhcUttaDVsRHNWRnk4WDRpYXRQbW81UllWenFweE9p
NEJNeXpPQWo5UktOVTRGNkFwTENhNVljNlFVMHViRTY2UUJzUkh5TWh0SHhQUTZOaFFTRk5LeXFKNGVn
VVZINDJlOFdBSmt1NlFZeHFicFRmZjEwdHVWdENwcENNMmQ5R1RCeHpseUY3aDZrQjBLMHRXNkF1a2Y0
QllNS3Jld2M5aUVGNGRUc1pPbEs0WVFEdkpxRjFOQ1BMMktXNjlnS212bXJmbTlZT2tHWE01MG9haTdp
R2l0amNzRHdkV3VBZEJ4OHJvek5tbnVQc2FCYVdPeVBWUjJBT0hVa1BrOW1jldwYTFDVXVOcU8xZzJk
RVExRXhBNFhaYUlnNlJLZjJvc2swMVJZektac0dKZEFUVnBaSkQzM2tnY2c4UDJnM0dYZFYzZnp4VTNH
bEtEQzhRUUlzQTJVYUJOODM4TWNiNmw3MUdZdGNuZlNDdGZFYlB0TU9OS2xxSejlhYWZHb3JaQzVMNFdw
TjZXRDVzZWlKelZtYWdrWUM2VThYY2dKWGpDSXJpR01lQjlobnY4RmFjNklDZnpROHFiZE5iZ1pTTTG9M
N0l0Tk4xZ1NmQ2JkeUE0RVFabHBGYlFEeVFFYUFJZUUycG9lbWRPU2x4a0FWYzBQU3kzZExEYWE= int
erface ethernet-1/1
/system/packet-trace-base64:
================================================================================
Ingress information for Packet 4 Ingress Interface ethernet-1/1
================================================================================
Type               : Bridged
Interface          : ethernet-1/1 (4401020001)
Net Instance       : macvrf1
================================================================================
================================================================================
Egress information for Packet 4
================================================================================
Interface          : Flooded in macvrf1
Egress Net Instance : macvrf1
================================================================================
```

6.1.3 Configuring the packet-trace tool (using pcap format)

Procedure

Use the following command to report the forwarding behavior for a specified test packet using packets specified in pcap format:

 # tools system packet-trace pcap-file *<file name>* **[interface** *<interface name>***] [max-packet-count** *<value>***] [packet-number** *<value>***]**

Packet trace command parameters for specifying pcap format are described in Table 4: Packet trace command parameters using pcap format.

Table 4: Packet trace command parameters using pcap format

Command/parameter	Description
tools system packet-trace	Reports the forwarding behavior for a specified test packet (file format)
pcap-file *<file name>*	Input file in pcap format
`interface` *`<interface name>`*	The name of the configured interface to inject the probe packet
max-packet-count *<value>*	Number of packets to read from the file (default: 100)
packet-number *<value>*	Use packet with the specified packet number from the pcap file

Example: packet-trace command using pcap format

```
# tools system packet-trace pcap-file data.pcap max-packet-count 1 packet-number 1
 interface ethernet-1/2
```

Example: output of packet trace in pcap format

```
+-------------------------------------------------------------------------------+
| Number     Time     Ingress   Source    Destina   Protoco   Length    Info    |
|                      port               tion       l                          |
+===============================================================================+
| 1       0.046971    etherne   192.1.7.1   10.1.5.1   UDP       2545   6722    |
|                                                                       \u2192  |
|                     t-1/2.1   1         8                             6789 Le |
|                                                                       n=2454  |
Enter packet number (default: [1]): 1
###[ Ethernet ]###
    dst       = 00:00:5e:00:53:41
    src       = 00:00:5e:00:53:d2
    type      = IPv4
###[ IP ]###
    version   = 4
    ihl       = 5
    tos       = 0x0
    len       = 2482
    id        = 0
    flags     =
    frag      = 0
    ttl       = 64
    proto     = udp
    chksum    = 0xd09
    src       = 192.0.2.1
    dst       = 10.1.5.1
    \options   \
```

```
###[ UDP ]###
    sport     = 6722
    dport     = smc_https
    len       = 2462
    chksum    = 0xcd6b
###[ Raw ]###
    load      =
'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\
x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_
`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x8
7\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\
x9a\x9b\x9c\x9d\x9e\x9f\xa0\....'
/system/packet-trace-base64:
    ================================================================================
    Ingress information for Packet 10 Ingress Interface ethernet-1/2
    ================================================================================
    Type                : Routed
    Interface           : ethernet-1/2 (4401020002)
    Sub interface       : ethernet-1/2.1
    Instance id         : 1
    Out Interface       : ethernet-1/1
    Nexthop ip          : 192.0.2.54
    ================================================================================
    ================================================================================
    Egress information for Packet 10 Egress Interface ethernet-1/1
    ================================================================================
    Interface           : ethernet-1/1 (4401020001)
    Sub interface       : ethernet-1/1.8
    Mac Address         : 00:00:5e:00:53:41
```

7 Mirroring

Mirroring copies IPv4 and IPv6 packets seen on a specified source, such as an interface (port) or subinterface (VLAN), or matching an ACL entry, and sends the packets to a specific destination, such as a locally attached traffic analyzer or a tunnel toward a remote destination.

By default, the mirrored packets include IPv4/IPv6 headers, as well as Ethernet headers. Traffic from multiple sources can be mirrored to a single destination, although traffic from a specific source cannot be mirrored to multiple destinations.

7.1 Mirror sources

The source for mirrored traffic can be an interface or subinterface or an ACL filter.

- Interfaces / subinterfaces

  A mirror source can be an interface, including all subinterfaces within that interface. The source can be a single interface (for example, `interface ethernet-1/1`) or a LAG (for example, `interface lag1`). Either a LAG member or LAG port can be mirrored. When a LAG port is configured as a mirror source, mirroring is enabled on all ports making up the LAG.

  The source can be a specific VLAN; that is, a subinterface within an interface where VLAN tagging is enabled (for example, `interface ethernet-1/1.1` or `lag1.1`).

  You can configure mirroring for traffic in a specific direction (ingress only, egress only) or bidirectional traffic (both ingress and egress).

  It is possible for the mirror source operational state to be down due to resource exhaustion. If the mirror source is not mirroring packets, check the operational state of the mirror sources .

- ACL filters

  A mirror source can be an IPv4 or IPv6 ACL filter, applied under one or more interfaces or subinterfaces. Traffic matching entries in the ingress ACL filter (regardless of whether the action is accept or drop), can be mirrored to the destination.

The following table lists hardware platform support for each mirror source.

Table 5: Hardware applicability (source mirroring)

Source	7220 IXR-D2/ D3	7220 IXR-D2L/ D3L	7220 IXR-D4/ D5	7250 IXR-6e/ 10e
Interface (ingress)	Yes	Yes	Yes	Yes
Interface (egress)	Yes	Yes	Yes	Yes
Subinterface (ingress)	Yes	Yes	Yes	Yes
Subinterface (egress)	Yes	Yes	No	Yes
ACL filter (ingress)	Yes	Yes	Yes	Yes

Source	7220 IXR-D2/ D3	7220 IXR-D2L/ D3L	7220 IXR-D4/ D5	7250 IXR-6e/ 10e
ACL filter (egress)	No	No	No	No

7.2 Mirror destinations

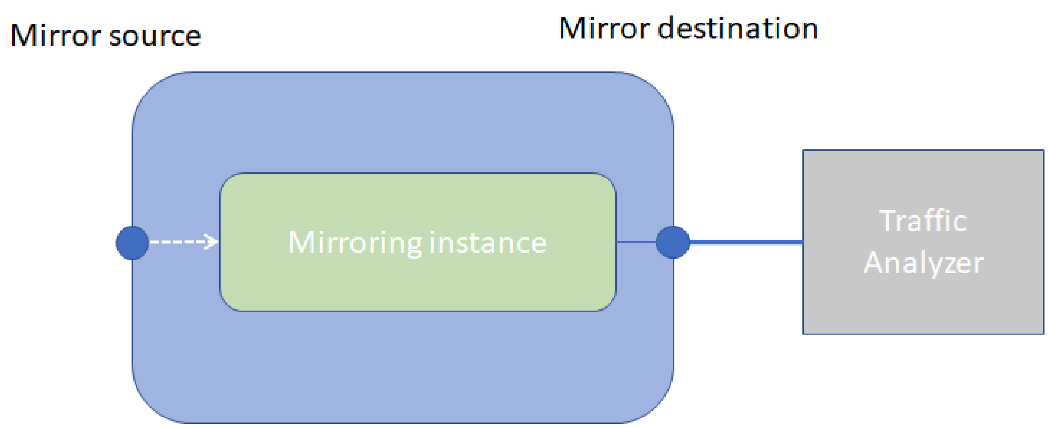
Traffic from the mirror source can be copied to a local destination (local mirroring) or encapsulated into a tunnel to a remote destination (remote mirroring).

Local mirroring

In a local mirroring configuration, both the mirror source and mirror destination reside on the same SR Linux node, as shown in Figure 2: Local mirroring.

In this configuration, the local destination is a Switched Port Analyzer (SPAN).

Figure 2: Local mirroring



For local mirroring, the following hardware types are supported:

- 7220 IXR-D2/D2L/D3/D3L
- 7220 IXR-D4/D5
- 7250 IXR-6e/10e

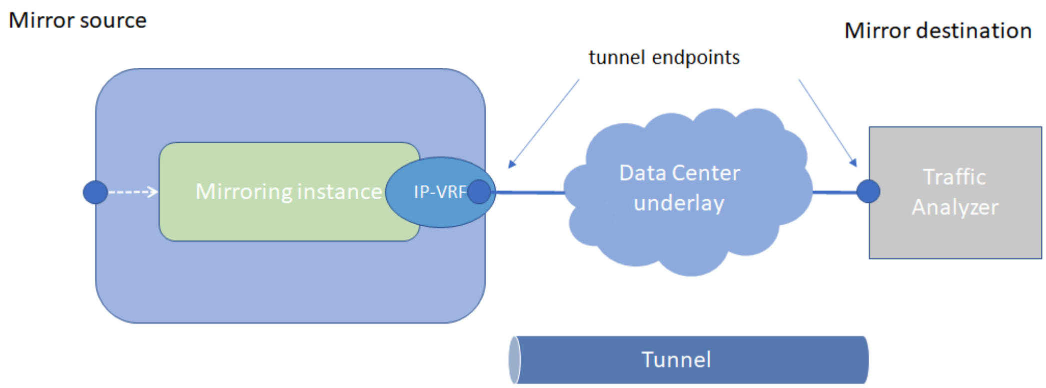
Remote mirroring

In a remote mirroring configuration, the mirror source and mirror destination are on different nodes. The mirror source resides on the SR Linux node, and the mirrored packets are encapsulated into a tunnel toward the mirror destination.

Figure 3: Remote mirroring shows a remote mirroring configuration. In this configuration, the remote destination is an Encapsulated Remote Switched Port Analyzer (ERSPAN).

Tunnel endpoints are defined within a specific network-instance, where the local tunnel endpoint IP address can be either a loopback subinterface address or any subinterface address within that network-instance.

Figure 3: Remote mirroring

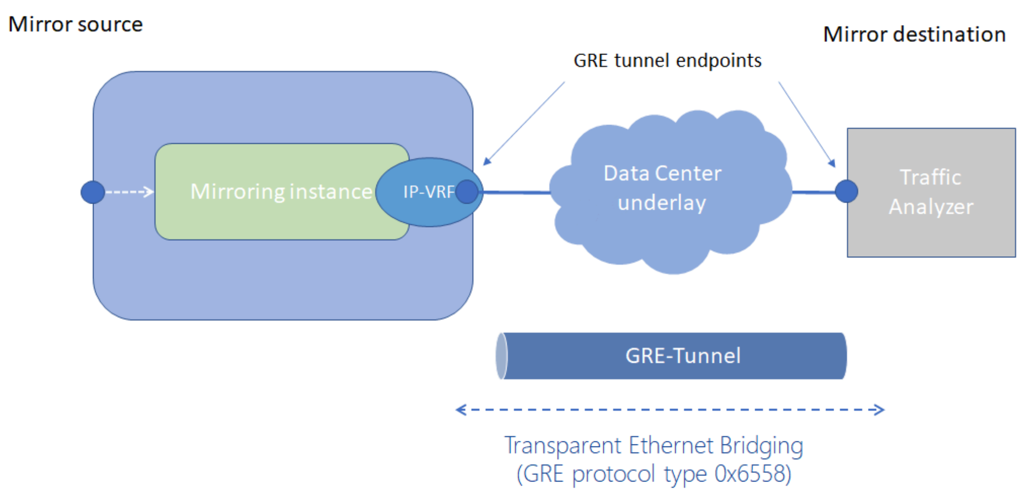


Mirroring to underlay (ERSPAN+GRE)

In a mirroring-to-underlay (ERSPAN+GRE) configuration, the mirrored packets, including IPv4/IPv6 header as well as Ethernet header, are tunneled using Transparent Ethernet Bridging (GRE protocol type 0x6558) or L3oGRE (protocol type 0x88be for 7250 IXR-6e/10e platforms only) toward the remote destination.

Figure 4: Mirroring to underlay shows a mirroring-to-underlay configuration.

Figure 4: Mirroring to underlay



For remote mirroring, the following hardware types are supported.

Table 6: Hardware applicability (destination mirroring - remote)

Destination	7220 IXR-D2/D3	7220 IXR-D2L/D3L	7220 IXR-D4/D5	7250 IXR-6e/10e
Underlay destination (GRE +ERSPAN) - IPv4 (ingress and egress	Yes	Yes	Yes	Yes

Destination	7220 IXR-D2/ D3	7220 IXR-D2L/ D3L	7220 IXR-D4/ D5	7250 IXR-6e/ 10e
Underlay destination (GRE +ERSPAN) - IPv6 (ingress-direction mirroring)	Yes	Yes	Yes	Yes
Underlay destination (GRE +ERSPAN) - IPv6 (egress-direction mirroring)	No	No	Yes	Yes

Note: For IXR-6e/10e, the remote encapsulation is L3oGRE. For all other platforms, the encapsulation is L2oGRE.

7.3 Configuring mirroring

To configure mirroring, you configure a mirroring-instance, which specifies the source and destination for the mirrored traffic. Multiple mirror sources can have a single destination, although traffic from a specific source cannot be mirrored to multiple destinations. Only one mirror destination can be configured per mirroring-instance. A mirror destination cannot be reused in multiple mirroring instances.

Within a mirroring-instance, if an interface is configured as mirror source, a subinterface within that interface cannot be added as another mirror source. If a LAG is defined as mirror destination, only the first 8 members of the LAG carry mirrored traffic. Note that on 7220 IXR-D4 and D5 platforms, a mirror destination port cannot be a LAG.

Mirrored traffic is considered Best Effort (BE) Forwarding Class.

7.3.1 Configuring mirroring sources

Procedure

To configure mirroring, you specify the source and destination for mirrored traffic within a mirroring-instance. The source in a mirroring-instance can be traffic on a specified interface, subinterface, or LAG, or can be packets matching an ACL entry.

Example: interface source

The following example shows a mirroring-instance configuration with an interface as the source for mirrored traffic:

```
--{ * candidate shared default }--[  ]--
# info system mirroring
    system {
        mirroring {
            mirroring-instance 1 {
                admin-state enable
                mirror-source {
                    interface ethernet-1/5 {
                        direction ingress-egress
                    }
                }
            }
        }
```

```
    }
```

Example: ACL source

The following example configures an ACL with an entry that matches TCP packets and applies the ACL to a subinterface. A mirroring-instance is configured that uses packets matching the ACL as the source for mirrored traffic.

```
--{ * candidate shared default }--[  ]--
# info acl acl-filter ip_tcp type ipv4
    acl {
        acl-filter ip_tcp type ipv4 {
            entry 1000 {
                description Match_TCP_Protocol
                match {
                    ipv4 {
                        protocol tcp
                    }
                }
                action {
                    accept {
                    }
                }
            }
        }
    }
```

```
--{ +* candidate shared default }--[  ]--
# info acl interface ethernet-1/1.1
    acl {
        interface ethernet-1/1.1 {
            interface-ref {
                interface ethernet-1/1
                subinterface 1
            }
            input {
                acl-filter ip_tcp type ipv4 {
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info system mirroring
    system {
        mirroring {
            mirroring-instance 1 {
                admin-state enable
                mirror-source {
                    acl {
                        acl-filter ip_tcp type ipv4 {
                            entry 1000 {
                            }
                        }
                    }
                }
            }
        }
    }
```

7.3.2 Configuring mirroring destinations

Procedure

In a mirroring-instance, you specify the destination for the mirrored traffic. The mirroring destination can be a local destination residing on the same SR Linux node as the mirroring source, or a remote destination where the mirrored traffic is sent via a tunnel. The tunneled traffic can be encapsulated with GRE protocol type 0x6558 or 0x88BE (7250 IXR-6e/10e platforms only).

Example: Local destination

The following example enables a subinterface to be a local mirror destination:

```
--{ * candidate shared default }--[  ]--
# info from running interface ethernet-1/4 subinterface 1
    interface ethernet-1/4 {
        subinterface 1 {
            type local-mirror-dest
            admin-state enable
            vlan {
                encap {
                    single-tagged {
                        vlan-id 1127
                    }
                }
            }
            local-mirror-destination {
                admin-state enable
            }
        }
    }
```

The following example configures a mirroring-instance where traffic from the mirror source is mirrored to the subinterface enabled as a local mirror destination:

```
--{ * candidate shared default }--[  ]--
# info system mirroring
    system {
        mirroring {
            mirroring-instance 1 {
                admin-state enable
                mirror-source {
                    interface ethernet-2/1 {
                        direction ingress-egress
                    }
                }
                mirror-destination {
                    local ethernet-1/4.1
                }
            }
        }
    }
```

Example: Remote destination using underlay

The following example configures a mirroring-instance that specifies the mirrored traffic be encapsulated into a tunnel within a network-instance. The mirrored traffic is encapsulated into a tunnel using L2oGRE to the remote destination.

```
--{ * candidate shared default }--[  ]--
# info system mirroring
    system {
        mirroring {
            mirroring-instance 1 {
                admin-state enable
                mirror-source {
                    interface ethernet-2/1 {
                        direction ingress-egress
                    }
                }
                mirror-destination {
                    remote {
                        encap l2ogre
                        network-instance IPVRF-1 {
                            tunnel-end-points {
                                src-ipv4 192.168.1.53
                                dst-ipv4 192.168.1.153
                            }
                        }
                    }
                }
            }
        }
    }
```

7.4 Displaying mirroring information

Procedure

Use the **info from state** command to display mirroring configuration information.

Example

```
--{ * candidate shared default }--[  ]--
# info from state system mirroring mirroring-instance 2
    system {
        mirroring {
            mirroring-instance 2 {
                admin-state enable
                oper-state down
                oper-down-reason local-mirror-subif-down
                mirror-source {
                    interface lag1 {
                        direction ingress-egress
                    }
                }
                mirror-destination {
                    local lag25.1
                }
            }
        }
```

```
    }
```

7.5 Displaying mirroring statistics

Procedure

On 7220 IXR-D2/D3 platforms, you can display the statistics per mirror destination interface using the **info from state interface * statistics** command. Refer to the **out-mirrored-packets** and the **out-mirrored-octets** fields.

On 7220 IXR-D4/D5 and 7250 IXR-6e/10e platforms, mirror destination statistics are not supported per-interface; it is only possible to display per-mirror-destination statistics. The statistics show the number of packets sent to the mirror destination.

On 7220 IXR-D4/D5 platforms, the statistics only include the number of packets mirrored in either the ingress or the egress direction. On 7250 IXR-6e/10e platforms, the statistics include the number of packets in the ingress direction and the number of octets mirrored in either the ingress or the egress direction.

The octet count for ERSPAN includes the GRE header (not just the actual mirror packet). The interfaces that egress the mirrored packet must adjust the MTU size to accommodate that additional GRE header. If the MTU size is smaller than the GRE packet, the mirrored packet is dropped.

There are no packet drop statistics for mirror destinations. The statistics represent all packets that have been successfully mirrored and sent to the mirror destination. It is possible for mirrored packets to be dropped due to over-congestion of multiple mirror sources to the same mirror destination. Mirrored packet drops can also occur because a mirror destination interface can be used for regular data traffic forwarding.

Example: Mirroring statistics on 7220 IXR-D2/D3 platforms

```
--{ running }--[  ]--
# info from state interface ethernet-1/48 statistics | filter fields out-mirror-octets
 out-mirror-packets
    interface ethernet-1/48 {
        statistics {
            out-mirror-octets 0
            out-mirror-packets 0
        }
```

Example: Mirroring statistics on 7220 IXR-D5 platform

```
--{ running }--[  ]--
# info from state system mirroring mirroring-instance * mirror-destination statistics
    system {
        mirroring {
            mirroring-instance eight {
                mirror-destination {
                    statistics {
                        ingress-mirrored-packets 22135
                        egress-mirrored-packets 22132
                    }
                }
            }
            mirroring-instance five {
                mirror-destination {
                    statistics {
                        ingress-mirrored-packets 6353567
```

```
                        egress-mirrored-packets 0
                    }
                }
            }
        }
    }
```

Example: Mirroring statistics on 7250 IXR-6e/10e platforms

```
--{ running }--[  ]--
# info from state system mirroring mirroring-instance ixia_one mirror-destination
 statistics
    system {
        mirroring {
            mirroring-instance ixia_one {
                mirror-destination {
                    statistics {
                        ingress-mirrored-packets 7417657
                        ingress-mirrored-octets 10384702600
                        egress-mirrored-octets 0
                    }
                }
            }
        }
    }
```

Customer document and product support

Customer documentation
Customer documentation welcome page

Technical support
Product support portal

Documentation feedback
Customer documentation feedback