# Nokia Service Router Linux
# 7215 Interconnect System
# 7220 Interconnect Router
# 7250 Interconnect Router
# 7730 Service Interconnect Router

Release 26.3

## ACL and Traffic Steering Guide

3HE 22264 AAAA TQZZA
Edition: 01
March 2026

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 ACLs and policy-based routing for the Nokia Service Router Linux (SR Linux). Examples of commonly used commands are provided.

This document is intended for network technicians, administrators, operators, service providers, and others who need to understand how the router is configured.

**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 Software Release Notes* for information on 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

The 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.
- 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.

The following table outlines platform grouping conventions used in the SR Linux documentation suite.

**Note:** Some platforms in the 7250 IXR support mixed systems. For more information about mixed system support, see "Chassis types" in the *Configuration Basics Guide*.

*Table 1: Platform grouping legend*

| Platform group | Description |
|---|---|
| 7215 IXS | 7215 IXS-A1 |
| 7220 IXR | All 7220 IXR platforms |
| 7220 IXR-D*x* | 7220 IXR-D1, 7220 IXR-D2, 7220 IXR-D2L, 7220 IXR-D3, 7220 IXR-D3L, 7220 IXR-D4, 7220 IXR-D5 |
| 7220 IXR-H*x* | 7220 IXR-H2, 7220 IXR-H3, 7220 IXR-H4, 7220 IXR-H4-32D, 7220 IXR-H5-32D, 7220 IXR-H5-64D, 7220 IXR-H5-64O |
| 7250 IXR[1] | 7250 IXR platforms |
| 7250 IXR Gen 2 | 7250 IXR-6, 7250 IXR-10 |
| 7250 IXR Gen 2c+ | 7250 IXR-6e with IMM2, 7250 IXR-10e with IMM2, 7250 IXR-X1b, 7250 IXR-X3b |
| 7250 IXR Gen 3 | 7250 IXR-6e with IMM3, 7250 IXR-10e with IMM3, 7250 IXR-18e, 7250 IXR-X4 |
| 7250 IXR-6e/10e (mixed system) | 7250 IXR-6e (mixed system)[2], 7250 IXR-10e (mixed system)[2] |
| 7730 SXR | 7730 SXR-1-32D, 7730 SXR-1d-32D, 7730 SXR-1x-44S |

## 1.3 Platform considerations

The SR Linux documentation supports multiple platforms, including 7730 SXR, 7220 IXR, 7250 IXR, and 7215 IXS. Most features described in the documentation work identically on all platforms that support SR Linux. However, some features may function differently based on the platform where SR Linux is running. For example, a feature may be supported on 7730 SXR, but not on other platforms, or there may be differences in how a feature works on 7730 SXR compared to other platforms.

- If a feature is exclusive to a specific platform, it is noted in the topic title or within the text of the topic.

1 References to the 7250 IXR platform group may be appended with (including mixed systems) or (excluding mixed systems) to indicate mixed system support.

2 References to this platform as part of 7250 IXR (mixed system) indicate mixed system support of 7250 IXR Gen 2c+ (IMM2) and 7250 IXR Gen 3 (IMM3). That is, the 7250 IXR-6e and 7250 IXR-10e can hold and support both IMM2 and IMM3 at the same time.

For example, Attaching an ACL to a subinterface (7730 SXR systems) applies only to the 7730 SXR platform.

- If a feature is supported on multiple platforms, but there are per-platform differences in how the feature works, these differences are described in the text.

  For example, Creating CPM filters provides configuration examples that apply to all supported platforms, as well as a configuration example that applies only to the 7730 SXR platform. In addition, the tables below summarize the per-platform differences.

- (7730 SXR platform only) If a feature has been verified as functioning on the 7730 SXR platform in the same way as the 7220 IXR/7250 IXR platform, it is noted in the topic.

  For example, Logging ACL resource usage applies equally to the 7730 SXR platform as it does to the7220 IXR/7250 IXR platform. The topic contains the note, "This feature is supported on both SXR and IXR platforms."

  This note does not imply that the feature is unsupported on other platforms.

### 7730 SXR platform considerations

The following table summarizes the considerations for ACL feature support on the 7730 SXR platform.

*Table 2: ACL feature support on 7730 SXR platforms*

| Feature | 7730 SXR considerations | See |
|---|---|---|
| IPv4/v6 interface filters | 7730 SXR allows up to two IPv4 ACLs and two IPv6 ACLs applied to input traffic on the same subinterface. | Attaching an ACL to a subinterface (7730 SXR systems) |
| ACL actions | Unsupported ACL actions on 7730 SXR:<br>• rate-limit policer in kbps (CPM filters)<br>ACL actions supported only on 7730 SXR:<br>• QoS forwarding-class<br>• QoS profile<br>• forward next-hop<br>• collect-stats | Supported ACL actions for 7730 SXR systems |
| CPM filters | A CPM filter can use a network-instance as a match condition. | Creating CPM filters |
| MAC filters | Unsupported on 7730 SXR | MAC ACLs |
| Packet capture filters | Unsupported on 7730 SXR | Packet capture filters |
| System filters | Unsupported on 7730 SXR | System filters |
| Rate limit action | The **entry-specific** parameter is set to `false` and cannot be configured. | Rate-limiting action for ACL filters |

| Feature | 7730 SXR considerations | See |
|---|---|---|
| | To use a separate policer instance for a given ACL or ACL entry, you must create a unique policer object. | |
| ACL statistics | The collect-stats ACL action is used for collecting statistics for ACL filter entries instead of the `statistics-per-entry` setting. | Collecting ACL statistics (7730 SXR systems) |
| Traffic steering | Traffic steering is configured using ACLs instead of policy-based forwarding (PBF) policies. | Traffic steering using ACLs |

# 2 What's new

This section lists the changes that were made in this release.

*Table 3: What's new in Release 26.3.1*

| Topic | Location |
|---|---|
| Support for MAC filters on bridged subinterfaces on 7250 IXR Gen 3 devices | Platform support for MAC interface filters |
| Policy forwarding to next-hop with and without VRF override | Actions for forwarding policies |
| Fallback actions for forwarding policies | Fallback actions for forwarding policies |
| Support for GRE and GUE tunnel decapsulation using forwarding policies | Using policy forwarding for tunnel decapsulation |
| Group-based policy ACLs for micro-segmentation | Group-based policy ACLs |

# 3 Access control lists

An Access Control List, or ACL, is an ordered set of rules that are evaluated on a packet-by-packet basis to determine whether access should be provided to a specific resource. ACLs can be used to drop unauthorized or suspicious packets from entering or leaving a routing device via specified interfaces.

An ACL filter is applied to a selected set of traffic contexts. A traffic context could be all the IPv4 or IPv6 packets arriving or leaving on a specific subinterface, or the out-of-band IP traffic arriving on a management interface, or all the in-band IPv4 or IPv6 packets that are locally terminating on the CPM of the router.

Each ACL filter rule, or entry, has a sequence ID. The ACL evaluates packets starting with the entry with the lowest sequence ID, progressing to the entry with the highest sequence ID. Evaluation stops at the first matching entry (that is, when the packet matches all of the conditions specified by the ACL entry).

SR Linux uses a common `acl-filter` object model to define ACLs by name and type for MAC, IPv4, and IPv6 traffic.

### IPv4/IPv6 ACLs

SR Linux supports `acl-filter` objects of type `ipv4` and `ipv6` for the following applications:

- Interface filters

  An interface filter is an IPv4 or IPv6 ACL that is applied to a routed or bridged subinterface to restrict traffic allowed to enter or exit the subinterface. An interface ACL can be applied to input or output traffic for one or more subinterfaces.

  See Interface filters for more information.

- CPM filters

  A CPM filter is an IPv4 or IPv6 ACL used for control plane protection. There is one CPM filter for IPv4 traffic and another CPM filter for IPv6 traffic. When an ingress IP packet is matched by a CPM filter rule, and it is a terminating packet (that is, it must be extracted to the CPM), then it is processed according to the matching CPM filter rule.

  See Control plane module (CPM) filters for more information.

- Packet capture filters

  A packet capture filter is an IPv4 or IPv6 ACL used to extract packets to the control plane for inspection by packet capture tools. When an ingress IP packet on any line card transits through the router, and it matches a rule in a packet capture filter policy, it is copied and extracted toward the CPM and delivered to a Linux virtual Ethernet interface so that it can be displayed by a packet capture utility, or encapsulated and forwarded to a remote monitoring system.

  The `acl-filter` name `capture` is reserved for packet capture filters.

  See Packet capture filters for more information.

- System filters (7220 IXR-D1/D2/D3/D4/D5 systems only)

  A system filter ACL is an IPv4 or IPv6 ACL that is evaluated early in the ingress pipeline, at a stage before tunnel termination occurs and before interface filters are run. For VXLAN traffic, system filters can match and drop unauthorized VXLAN tunnel packets before they are decapsulated, based on information in the outer header.

The `acl-filter` name `system` is reserved for system filters.

See System filters for more information.

### MAC ACLs

SR Linux supports a Layer 2 traffic `acl-filter` object of type `mac` for the following applications:

- Interface MAC filters

  An interface MAC filter is a Layer 2 ACL that is applied to a routed or bridged subinterface to restrict the Ethernet frames allowed to enter or exit from that subinterface. An interface MAC filter can match Ethernet frames carrying an IP or non-IP payload and can be applied to the input or output traffic of one or more subinterfaces.

  See Interface filters for more information.

- CPM filter MAC ACLs

  A CPM filter MAC ACL is used for control plane protection. When a CPM filter MAC ACL is created, its rules are automatically evaluated against all non-IP traffic that is extracted to the CPM as a result of a match with an extraction rule (for example, a match of a specific well-known MAC DA value). This is regardless of whether the traffic entered from of network instance, subinterface, TD3 pipeline, and so on.

  See Control plane module (CPM) filters for more information.

MAC ACLs are not supported on 7730 SXR systems.

## 3.1 ACL actions

When a packet matches an ACL entry, an action specified by the ACL entry is applied to the packet.

For traffic transiting through the router, ACL entries support the following actions:

- accept – Allow the packet through to the next processing function.
- drop – Discard the packet without ICMP generation.
- log – Send information about the packet to the log application.
- rate-limit policer – The packet is allowed through to the next processing function and rate-limited by a policer instance.

For traffic terminating on the CPM of the router, the following additional action is supported:

- rate-limit system-cpu-policer – If the packet has been extracted to the CPM, feed the packet to a software-based policer, which determines if the packet should be delivered to the CPM application or dropped because a packet-per-second rate is exceeded.

On 7730 SXR systems, the following additional actions are supported:

- forward next-hop – Forward the packet to a specified next-hop address.
- forwarding-class – Assign the packet to a QoS forwarding class
- profile – Assign a QoS profile to the packet
- collect-stats – Record statistics for matching packets at the filter-entry level. Statistics are aggregated for all subinterfaces using the same filter entry. Statistics for matching input and output packets are recorded separately .

If a packet matches an ACL entry, no further evaluation is done for the packet. If the packet does not match any ACL entry, the default action is accept. To drop traffic that does not match any ACL entry, you can optionally configure an entry with the highest sequence ID in the ACL to drop all traffic. This causes traffic that does not match any of the lower-sequence ACL entries to be dropped.

The supported actions for each type of ACL differ based on the hardware platform where the ACL is configured. The following tables indicate which actions are supported for each ACL filter type for each hardware platform.

### 3.1.1 Supported ACL actions for 7730 SXR systems

The following table lists the supported actions for each ACL filter type on 7730 SXR systems.

*Table 4: Supported actions for each ACL filter type (7730 SXR)*

| ACL filter type | Action | Supported? |
|---|---|---|
| IPv4/IPv6 Interface filter (input) | accept | Yes |
| | log | Yes |
| | rate-limit policer (kbps) | Yes |
| | forward next-hop | Yes |
| | forwarding-class | Yes |
| | profile | Yes |
| | drop | Yes |
| | collect-stats | Yes |
| IPv4/IPv6 Interface filter (output) | accept | Yes |
| | log | Yes |
| | rate-limit policer | No |
| | forward next-hop | No |
| | forwarding-class | Yes |
| | profile | Yes |
| | drop | Yes |
| | collect-stats | Yes |
| IPv4/IPv6 CPM filter | accept | Yes |
| | log | Yes |
| | accept + rate-limit policer (pps) | Yes |

| ACL filter type | Action | Supported? |
|---|---|---|
| | accept + rate-limit system-cpu-policer | Yes |
| | drop | Yes |
| | collect-stats | Yes |
| Packet capture filter | accept | No |
| | copy | No |
| System filter | accept | No |
| | drop | No |
| | drop + log | No |
| Interface MAC filter (input) | accept | No |
| | accept + log | No |
| | drop | No |
| | drop + log | No |
| Interface MAC filter (output) | accept | No |
| | accept + log | No |
| | drop | No |
| | drop + log | No |
| CPM filter MAC ACL | accept | No |
| | accept + log | No |
| | accept + rate-limit policer | No |
| | accept + rate-limit system-cpu-policer | No |
| | drop | No |
| | drop + log | No |

### 3.1.2 Supported ACL actions for 7250 IXR systems

The following table lists the supported actions for each ACL filter type on 7250 IXR systems.

*Table 5: Supported actions for each ACL filter type (7250 IXR)*

| ACL filter type | Action | Supported? |
|---|---|---|
| IPv4/IPv6 Interface filter (input) | accept | Yes |
| | accept + log | Yes |
| | drop | Yes |
| | drop + log | Yes |
| IPv4/IPv6 Interface filter (output) | accept | Yes |
| | accept + log | Yes |
| | drop | Yes |
| | drop + log | Yes |
| IPv4/IPv6 CPM filter | accept | Yes |
| | accept + log | Yes |
| | accept + rate-limit policer | Yes |
| | accept + rate-limit system-cpu-policer | Yes |
| | drop | Yes |
| | drop + log | Yes |
| Packet capture filter | accept | Yes |
| | copy | Yes |
| System filter | accept | No |
| | drop | No |
| | drop + log | No |
| Interface MAC filter (input) | accept | Yes |
| | accept + log | Yes |
| | drop | Yes |
| | drop + log | Yes |
| | accept + rate-limit policer | No |
| Interface MAC filter (output) | accept | Yes |
| | accept + log | Yes |

| ACL filter type | Action | Supported? |
|---|---|---|
| | drop | Yes |
| | drop + log | Yes |
| CPM filter MAC ACL | accept | No |
| | accept + log | No |
| | accept + rate-limit policer | No |
| | accept + rate-limit system-cpu-policer | No |
| | drop | No |
| | drop + log | No |

### 3.1.3 Supported ACL actions for 7220 IXR-D1/D2/D3 systems

The following table lists the supported actions for each ACL filter type on 7220 IXR-D1, D2, and D3 systems.

*Table 6: Supported actions for each ACL filter type (7220 IXR-D1, D2, and D3)*

| ACL filter type | Action | Supported? |
|---|---|---|
| IPv4/IPv6 Interface filter (input) | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | drop | Yes |
| | drop + log | Yes |
| IPv4/IPv6 Interface filter (output) | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | drop | Yes |
| | drop + log | No log generated |
| IPv4/IPv6 CPM filter | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | accept + rate-limit system-cpu-policer | Yes |

| ACL filter type | Action | Supported? |
|---|---|---|
| | drop | Yes |
| | drop + log | Yes |
| Packet capture filter | accept | Yes |
| | copy | Yes |
| System filter | accept | Yes |
| | drop | Yes |
| | drop + log | Yes |
| Interface MAC filter (input) | accept | Yes |
| | accept + log | No log generated |
| | drop | Yes |
| | drop + log | Yes |
| Interface MAC filter (output) | accept | Yes |
| | accept + log | No log generated |
| | drop | Yes |
| | drop + log | No log generated |
| CPM filter MAC ACL | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | accept + rate-limit system-cpu-policer | Yes |
| | drop | Yes |
| | drop + log | Yes |

### 3.1.4 Supported ACL actions for 7220 IXR-D4/D5 systems

The following table lists the supported actions for each ACL filter type on 7220 IXR-D4 and 7220 IXR-D5 systems.

*Table 7: Supported actions for each ACL filter type (7220 IXR-D4 and 7220 IXR-D5)*

| ACL filter type | Action | Supported? |
|---|---|---|
| IPv4/IPv6 Interface filter (input) | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | drop | Yes |
| | drop + log | Yes |
| IPv4/IPv6 Interface filter (output) | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | drop | Yes |
| | drop + log | No log generated |
| IPv4/IPv6 CPM filter | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | No |
| | accept + rate-limit system-cpu-policer | Yes |
| | drop | Yes |
| | drop + log | No log generated |
| Packet capture filter | accept | Yes |
| | copy | Yes |
| System filter | accept | Yes |
| | drop | Yes |
| | drop + log | Yes |
| Interface MAC filter (input) | accept | Yes |
| | accept + log | No log generated |
| | drop | Yes |
| | drop + log | Yes |

| ACL filter type | Action | Supported? |
|---|---|---|
| Interface MAC filter (output) | accept | Yes |
| | accept + log | No log generated |
| | drop | Yes |
| | drop + log | No log generated |
| CPM filter MAC ACL | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | accept + rate-limit system-cpu-policer | Yes |
| | drop | Yes |
| | drop + log | Yes |

### 3.1.5 Supported ACL actions for 7220 IXR-H systems

The following table lists the supported actions for each ACL filter type on 7220 IXR-H2, H3, H4, and H5 systems.

*Table 8: Supported actions for each ACL filter type (7220 IXR-H)*

| ACL filter type | Action | Supported? |
|---|---|---|
| IPv4/IPv6 Interface filter (input) | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes (7220 IXR-H4 only) |
| | drop | Yes |
| | drop + log | Yes |
| IPv4/IPv6 Interface filter (output) | accept | Yes |
| | accept + log | No log generated |
| | drop | Yes |
| | drop + log | No log generated |
| IPv4/IPv6 CPM filter | accept | Yes |
| | accept + log | No log generated |

| ACL filter type | Action | Supported? |
|---|---|---|
| | accept + rate-limit policer | No |
| | accept + rate-limit system-cpu-policer | Yes |
| | drop | Yes |
| | drop + log | No log generated |
| Packet capture filter | accept | Yes |
| | copy | Yes |
| System filter | accept | No |
| | drop | No |
| | drop + log | No |
| Interface MAC filter (input) | accept | No |
| | accept + log | No |
| | drop | No |
| | drop + log | No |
| Interface MAC filter (output) | accept | No |
| | accept + log | No |
| | drop | No |
| | drop + log | No |
| CPM filter MAC ACL | accept | No |
| | accept + log | No |
| | accept + rate-limit policer | No |
| | accept + rate-limit system-cpu-policer | No |
| | drop | No |
| | drop + log | No |

### 3.1.6 Supported ACL actions for 7215 IXS systems

The following table lists the supported actions for each ACL filter type on 7215 IXS systems.

*Table 9: Supported actions for each ACL filter type (7215 IXS)*

| ACL filter type | Action | Supported? |
|---|---|---|
| IPv4/IPv6 Interface filter (input) | accept | Yes |
| | accept + log | No log generated |
| | drop | Yes |
| | drop + log | Yes |
| IPv4/IPv6 Interface filter (output) | accept | No |
| | accept + log | No |
| | drop | No |
| | drop + log | No |
| IPv4/IPv6 CPM filter | accept | Yes |
| | accept + log | No log generated |
| | accept + rate-limit policer | Yes |
| | accept + rate-limit system-cpu-policer | Yes |
| | drop | Yes |
| | drop + log | Yes |
| Packet capture filter | accept | Yes |
| | copy | Yes |
| System filter | accept | No |
| | drop | No |
| | drop + log | No |
| Interface MAC filter (input) | accept | Yes |
| | accept + log | No |
| | drop | Yes |
| | drop + log | Yes |
| Interface MAC filter (output) | accept | No |
| | accept + log | No |
| | drop | No |

| ACL filter type | Action | Supported? |
|---|---|---|
| | drop + log | No |
| CPM filter MAC ACL | accept | No |
| | accept + log | No |
| | accept + rate-limit policer | No |
| | accept + rate-limit system-cpu-policer | No |
| | drop | No |
| | drop + log | No |

## 3.2 ACL match conditions

You can specify the following match conditions in IPv4, IPv6, and MAC ACLs.

### Match conditions for IPv4 ACLs

IPv4 ACLs analyze IPv4 packets. The following match criteria are supported by IPv4 ACLs:

- IPv4 destination prefix and prefix-length
- IPv4 destination address and address-mask
- TCP/UDP destination port (range)
- ICMP type/code
- IP protocol number
- IPv4 source prefix and prefix-length
- IPv4 source address and address-mask
- TCP/UDP source port (range)
- TCP flags: RST, SYN, and ACK
- Packet fragmentation: whether the packet is a fragment
- Packet fragmentation: whether the packet is a first-fragment (fragment-offset=0 and more-fragments=1)
- IP DSCP
- IPv4 source prefix list
- IPv4 destination prefix list
- Network-instance for CPM filters (7730 SXR systems)
- IP option: true (at least one option is present) or false (no option is present) (7730 SXR systems)
- IPv4 TTL value or range (7730 SXR systems)

### Match conditions for IPv6 ACLs

IPv6 ACLs analyze IPv6 packets. The following match criteria are supported by IPv6 ACLs:

- IPv6 destination prefix and prefix-length
- IPv6 destination address and address-mask
- TCP/UDP destination port (range)
- ICMPv6 type/code
- IPv6 next-header value. This is the value in the very first next-header field, in the fixed header.
- IPv6 source prefix and prefix-length
- IPv6 source address and address-mask
- TCP/UDP source port (range)
- TCP flags: RST, SYN, and ACK
- IP DSCP
- IPv6 source prefix list
- IPv6 destination prefix list
- Network-instance for CPM filters (7730 SXR systems)
- Hop-limit value or range (7730 SXR systems)

### Match conditions for MAC ACLs

MAC ACLs analyze Ethernet frames. The following match criteria are supported by MAC ACLs:

- MAC destination address with configurable address mask
- MAC source address match with configurable address mask
- outermost VLAN ID (single VLAN ID value or one contiguous VLAN ID range)
- Ethertype number after the last 802.1Q VLAN tag (if any), specified as a number or a well-known name

## 3.3 Interface filters

An interface filter is an IPv4 or IPv6 ACL that restricts traffic allowed to enter or exit a subinterface. IPv4 and IPv6 ACLs can be applied to a subinterface to restrict IP traffic entering or exiting that subinterface.

A MAC interface filter is a Layer 2 ACL that can be applied to a routed or bridged subinterface to restrict the Ethernet frames allowed to enter or exit that subinterface. An interface MAC ACL can match Ethernet frames carrying an IP or non-IP payload.

The following sections provide details on the traffic scope of input and output IPv4/IPv6 ACLs, as well as MAC interface filters.

### Input IPv4/IPv6 ACLs

When a routed subinterface of an Ethernet port or LAG has an input IPv4 or IPv6 ACL, the ACL rules apply to all packets matched by the subinterface encapsulation (untagged or single-tagged). The rules do not apply to MPLS LSR packets; MPLS packets are forwarded based on label lookup (top label or not).

At the network ingress of an eLER:

- On IXR Gen 2, IXR Gen 2c+, and IXR Gen 3 platforms, the ACL rules apply to non-tunneled packets only; tunnels that terminate in the default network-instance or other VRFs are excluded.

- On 7730 SXR platforms, for tunneled packets that end up in the base network-instance, the ACL rules apply to received MPLS packets forwarded based on IP FIB lookup (after popping one or more labels and regardless of the network-instance where the IP route is found).

When a bridged subinterface of an Ethernet port or LAG has an input IPv4/IPv6 ACL, the rules apply to all IPv4 packets matched by the subinterface encapsulation (untagged or single-tagged), including IPv4/IPv6 switched frames and IPv4 packets forwarded to the IRB.

When an IRB subinterface has an input IPv4/IPv6 ACL, the rules apply to packets that ingress via the bridged subinterfaces of the MAC VRF, terminate at the IRB (where the destination MAC is the IRB MAC), and then are forwarded within the default network-instance or IP VRF network-instance. If a packet enters a bridged subinterface and is forwarded through the IRB subinterface, it may be matched by both a rule of the ACL applied to the bridged subinterface and a rule of the ACL applied to the IRB subinterface; in this situation the IRB rule takes precedence.

### Output IPv4/IPv6 ACLs

When a routed subinterface of an Ethernet port or LAG has an output IPv4 or IPv6 ACL, the ACL rules apply to most packets forwarded out of the subinterface, regardless of egress encapsulation (untagged or single-tagged). Note the following exclusions:

- VXLAN-encapsulated packets bypass the ACLs; no action is taken even if there is a match of the inner IP/L4 headers or the outer IP header.
- MPLS-encapsulated packets bypass the ACLs.

When a bridged subinterface of an Ethernet port or LAG has an output IPv4 or IPv6 ACL, the rules apply only to IPv4/IPv6 switched frames that come from another bridged subinterface of the MAC-VRF, regardless of egress encapsulation (untagged or single-tagged).

When an IRB subinterface has an output IPv4 or IPv6 ACL, the rules apply to IPv4/IPv6 packets that have the IRB subinterface as a next-hop (where the source MAC is the IRB MAC) and that subsequently get forwarded within the MAC-VRF network-instance.

### Platform support for IPv4/IPv6 ACLs

The following table summarizes support for IPv4/IPv6 ACLs for each subinterface type on SR Linux platforms.

*Table 10: Platform support for IPv4/IPv6 ACLs on routed/bridged/IRB subinterfaces*

| Subinterface type | 7220 IXR-Dx | 7220 IXR-Hx | 7250 IXR | 7215 IXS | 7730 SXR |
|---|---|---|---|---|---|
| Bridged | Yes | 7220 IXR-H2, 7220 IXR-H3: No<br>7220 IXR-H4, 7220 IXR-H5: Yes | No | Yes, ingress only | Yes |
| Routed | Yes | Yes | Yes | Yes, ingress only | Yes |

| Subinterface type | 7220 IXR-Dx | 7220 IXR-Hx | 7250 IXR | 7215 IXS | 7730 SXR |
|---|---|---|---|---|---|
| IRB | Yes | 7220 IXR-H2, 7220 IXR-H3: No<br>7220 IXR-H4, 7220 IXR-H5: Yes | Yes, ingress only | Yes, ingress only | Yes |

### MAC interface filters

A MAC interface filter is a Layer 2 ACL that can be applied to a routed or bridged subinterface to restrict the Ethernet frames allowed to enter or exit that subinterface. An interface MAC ACL can match Ethernet frames carrying an IP or non-IP payload.

For a specific direction of traffic (input or output), a routed or bridged subinterface of an Ethernet port or LAG can have either a MAC interface ACL or an IPv4/IPv6 interface ACL applied, but not both at the same time. This restriction also applies to subinterfaces of the `mgmt0` management interface.

*Table 11: MAC ACL filter rules*

| Subinterface type | MAC ACL traffic type | Rules |
|---|---|---|
| Routed | Input | Rules apply to all Ethernet frames matched by the subinterface encapsulation, even if they contain an IP or MPLS-IP payload. |
| | Output | Rules apply to all Ethernet frames egressing the subinterface except VXLAN-encapsulated packets and MPLS-encapsulated packets. |
| Bridged | Input | Rules apply to all Ethernet frames matched by the subinterface encapsulation, even if they contain an IP or MPLS-IP payload, and regardless of whether they are switched or forwarded to the IRB. However, if the IRB also has an input IPv4/IPv6 ACL applied to it, the IRB action takes priority over the bridged subinterface MAC ACL action. |
| | Output | Rules apply to all Ethernet frames egressing the subinterface except frames routed from the IRB subinterface. |

ACLs are not supported on loopback or `system0` subinterfaces.

### Platform support for MAC interface filters

The following table summarizes support for MAC interface filters for each subinterface type on SR Linux platforms.

*Table 12: Platform support for MAC interface filters*

| Subinterface type | 7220 IXR-Dx | 7220 IXR-Hx | 7250 IXR Gen 2/2c+ | 7250 IXR Gen 3 | 7215 IXS | 7730 SXR |
|---|---|---|---|---|---|---|
| Bridged | Yes | No | Yes, no support for VLAN match on egress | Yes, no support for VLAN match on egress | Yes, ingress only | No |
| Routed | Yes | No | No | No | Yes, ingress only | No |
| IRB | Yes | No | No | No | Yes, ingress only | No |

## 3.3.1 Creating an IPv4 ACL

### Procedure

To configure an IPv4 ACL filter, you create an `acl-filter` of type `ipv4` and specify one or more entries consisting of match conditions and the action to take for IPv4 traffic that matches the conditions.

### Example: Accept IPv4 traffic matching specified IP address and source/destination port

The following is an example IPv4 ACL that has one entry. This example creates an IPv4 ACL named `ip_tcp`. Within the `ip_tcp` ACL, an entry with sequence ID 1000 is configured. The action is specified as `accept`, with logging set to `true`.

The filter matches packets with IP destination address 10.1.3.1/32; for TCP traffic, if the source address 10.1.5.1/32, destination port 6789, and source port 6722 matches the filter, the traffic stream is accepted.

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter ip_tcp type ipv4 {
            entry 1000 {
                description "Match IP Address TCP Protocol Ports"
                match {
                    ipv4 {
                        protocol tcp
                        destination-ip {
                            prefix 10.1.3.1/32
                        }
                        source-ip {
                            prefix 10.1.5.1/32
                        }
                    }
                    transport {
```

```
                        destination-port {
                            value 6789
                        }
                        source-port {
                            value 6722
                        }
                    }
                }
                action {
                    log true
                    accept {
                    }
                }
            }
        }
    }
```

### Example: Drop all matching traffic

The following is an example of an entry added to the `ip_tcp` ACL that causes all traffic to be dropped. Because it has the highest sequence ID, traffic that matches any of the lower-sequenced ACL entries would have been accepted before being evaluated by this entry. Only traffic that did not match any of the other ACL entries would be dropped by this entry.

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter ip_tcp type ipv4 {
            entry 65535 {
                action {
                    log true
                    drop {
                    }
                }
            }
        }
    }
```

Note that the `drop` action with logging set to `true` is not supported on 7220 IXR-D*x*, 7220 IXR-H*x*, and 7215 IXS-A1 systems when it is attached as an egress filter.

### Example: Match based on IPv4 address and mask

The following example uses an address mask to specify the source IPv4 address. The address mask, entered in IPv4 address format such as 0.0.0.255, indicates the bit values that must be matched exactly (0 in the mask) and the bit values that act as a wildcard (255 in the mask).

A packet matches if its IP address, logically ANDed with the inverse of the mask, results in the same value as the logical AND of the configured IP address and inversed mask.

In the example, the match condition is `address 10.10.10.0` and `mask 0.0.0.255`. This is the equivalent of configuring the match condition `prefix 10.10.10.0/24`.

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter ip_tcp type ipv4 {
            entry 1 {
                match {
                    ipv4 {
                        destination-ip {
```

```
                            address 10.10.10.0
                            mask 0.0.0.255
                        }
                    }
                }
                action {
                    log true
                    drop {
                    }
                }
            }
        }
    }
```

### Example: Match based on IPv4 prefix list

The following example matches based on the IPv4 prefixes configured in a prefix list. In this example, an IPv4 prefix list `p1` is configured under `match-list`. This IPv4 prefix list is specified as the source IP match condition in an ACL filter entry. If the source address of a packet matches one of the prefixes in the list, the packet is dropped.

```
--{ * candidate shared default }--[  ]--
# info with-context acl match-list ipv4-prefix-list pl1
    acl {
        match-list {
            ipv4-prefix-list pl1 {
                description "IPv4 prefix match list"
                prefix 10.10.0.0/16 {
                }
                prefix 10.20.0.0/16 {
                }
                prefix 10.30.0.0/16 {
                }
            }
        }
    }
```

```
--{ +* candidate shared default }--[  ]--
# info with-context acl acl-filter pfilter type ipv4
    acl {
        acl-filter pfilter type ipv4 {
            entry 100 {
                match {
                    ipv4 {
                        source-ip {
                            prefix-list pl1 {
                            }
                        }
                    }
                }
                action {
                    drop {
                    }
                }
            }
        }
    }
```

## 3.3.2 Creating an IPv6 ACL

### Procedure

To configure an IPv6 ACL filter, you create an `acl-filter` of type `ipv6` and specify one or more entries consisting of match conditions and the action to take for IPv6 traffic that matches the conditions.

### Example: Accept IPv6 traffic matching specified conditions

The following is an example IPv6 ACL that has one entry. This example creates an IPv6 ACL named `ipv6_tcp`. Within the `ipv6_tcp` ACL, an entry with sequence ID 100 is configured. The action is specified as `accept`, with logging set to `true`.

The filter matches packets with IPv6 destination address 2001:db8:1:3::1/120; for TCP traffic, if the source address 2001:db8:1:5::1/120, destination port 6789, and source port 6722 matches the filter, the traffic stream is accepted.

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter ipv6_tcp type ipv6 {
            entry 100 {
                description "Match Dest Address TCP Src Address DP SP"
                match {
                    ipv6 {
                        next-header tcp
                        destination-ip {
                            prefix 2001:db8:1:3::/120
                        }
                        source-ip {
                            prefix 2001:db8:1:5::/120
                        }
                    }
                    transport {
                        destination-port {
                            value 6789
                        }
                        source-port {
                            value 6722
                        }
                    }
                }
                action {
                    log true
                    accept {
                    }
                }
            }
        }
    }
```

### Example: Match based on IPv6 prefix list

The following example matches based on the IPv6 prefixes configured in a prefix list. In this example, an IPv6 prefix list p2 is configured under `match-list`. This IPv6 prefix list is specified as the destination IP match condition in an ACL filter entry. If the destination address of a packet matches one of the prefixes in the list, the packet is dropped.

```
--{ * candidate shared default }--[  ]--
```

```
# info with-context acl match-list ipv6-prefix-list p2
    acl {
        match-list {
            ipv6-prefix-list p2 {
                prefix 2001:db8:1:3::/120 {
                }
                prefix 2001:db8:1:5::/120 {
                }
                prefix 2001:db8:1:7::/120 {
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context acl acl-filter pfilter2 type ipv6
    acl {
        acl-filter pfilter2 type ipv6 {
            entry 100 {
                match {
                    ipv6 {
                        destination-ip {
                            prefix-list p2 {
                            }
                        }
                    }
                }
                action {
                    drop {
                    }
                }
            }
        }
    }
```

### 3.3.3 Creating a MAC ACL

#### Procedure

To create a MAC ACL specify statements consisting of match criteria (see Match conditions for MAC ACLs) and actions (see Supported ACL actions for 7220 IXR-D1/D2/D3 systems).

**Note:** MAC ACLs are not supported on 7730 SXR systems.

#### Example: Drop traffic from a source MAC

This example creates a MAC ACL named `mac01`. Within the `mac01` ACL, an entry with sequence ID 100 is configured. The filter matches Ethernet frames with a source MAC address of 00:00:5E:00:53:01. The action is specified as `drop`, with logging set to `true`.

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter mac01 type mac {
            entry 100 {
                match {
                    l2 {
                        source-mac {
```

```
                            address 00:00:5E:00:53:01
                        }
                    }
                }
                action {
                    log true
                    drop {
                    }
                }
            }
        }
    }
```

### Example: Accept traffic using MAC address mask

The following example uses an address mask to specify the source MAC address. The address mask, entered in MAC address format such as ff:ff:ff:ff:00:00, indicates the bit values that must be matched exactly (FF value in the mask) and the bit values that can be 0 or 1 (00 value in the mask).

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter mac01 type mac {
            entry 200 {
                match {
                    l2 {
                        source-mac {
                            address 00:00:5E:00:53:01
                            mask FF:FF:FF:FF:00:00
                        }
                    }
                }
                action {
                    accept {
                    }
                }
            }
        }
    }
```

### Example: Drop traffic of specific Ethertype

The following example drops Ethernet frames with ARP Ethertype (0x0806).

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter mac01 type mac {
            entry 300 {
                match {
                    l2 {
                        ethertype arp
                    }
                }
                action {
                    log true
                    drop {
                    }
                }
            }
        }
```

```
    }
```

### Example: Drop traffic with specific outermost VLAN ID

You can configure a specific VLAN ID or range of VLAN IDs as match criteria. The match is based on the outermost VLAN ID of the Ethernet frame. The VLAN ID can be specified as an integer from 0 to 4095 or **none**. Specifying 0 as the VLAN ID matches priority-tagged frames; specifying **none** matches untagged frames.

The following example drops Ethernet frames with VLAN ID 4095.

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter mac01 type mac {
            entry 400 {
                match {
                    l2 {
                        vlan {
                            outermost-vlan-id {
                                value 4095
                            }
                        }
                    }
                }
                action {
                    log true
                    drop {
                    }
                }
            }
        }
    }
```

The following example configures a range of VLAN IDs, 100 to 999 inclusive, as match criteria.

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        acl-filter mac01 type mac {
            entry 500 {
                match {
                    l2 {
                        vlan {
                            outermost-vlan-id {
                                range {
                                    start 100
                                    end 999
                                }
                            }
                        }
                    }
                }
                action {
                    log true
                    drop {
                    }
                }
            }
        }
    }
```

Note the following when specifying VLAN IDs as match criteria:

- When used in an ingress ACL applied to a routed or bridged subinterface, the VLAN ID is matched before the subinterface-defining VLAN tag (of a single-tagged subinterface) has been removed.
- When used in an egress ACL applied to a routed subinterface, the VLAN ID is matched after the subinterface-defining VLAN tag (of a single-tagged subinterface) has been added.
- When used in an egress ACL applied to a bridged subinterface the VLAN ID is matched before the subinterface defining VLAN tag (of a single-tagged subinterface) has been added.

### 3.3.4 Attaching an ACL to a subinterface

#### Procedure

After an ACL is configured, you can attach it to a subinterface so that traffic entering or exiting the subinterface is subject to the rules defined in the ACL. For an interface ACL to have an effect on the system, it must be explicitly applied to the input and, or output traffic of at least one subinterface.

The interface reference (`interface-ref`) binding to the subinterface is created as part of the ACL configuration context, along with the `input` and `output acl-filter` attachment.

#### Example: Attach IPv4/IPv6 ACLs to a subinterface

The following example applies IPv4 and IPv6 ACLs to inbound traffic on a subinterface.

```
--{ * candidate shared default }--[  ]--
# info with-context acl interface ethernet-1/16.1
    acl {
        interface ethernet-1/16.1 {
            interface-ref {
                interface ethernet-1/16
                subinterface 1
            }
            input {
                acl-filter ipv4-filter type ipv4 {
                }
                acl-filter ipv6-filter type ipv6 {
                }
            }
        }
    }
```

#### Example: Attach a MAC ACL to a subinterface

The following example applies a MAC ACL to outbound traffic on a subinterface.

To apply a MAC ACL to traffic in the outbound direction on any subinterface, you must set the **acl egress-mac-filtering** command to **true**. (Applicable to 7220 IXR-D*x* systems only)

```
--{ * candidate shared default }--[  ]--
# info with-context acl interface ethernet-1/16.1
    acl {
        interface ethernet-1/16.1 {
            interface-ref {
                interface ethernet-1/16
                subinterface 1
            }
            output {
```

```
                acl-filter mac01 type mac {
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context acl
    acl {
        egress-mac-filtering true
    }
```

### 3.3.5 Attaching an ACL to a subinterface (7730 SXR systems)

7730 SXR systems allow the following number of interface ACLs to be applied to input or output traffic on a subinterface:

Input traffic:

- up to two IPv4 ACLs
- up to two IPv6 ACLs

Output traffic:

- one IPv4 ACL
- one IPv6 ACL

The system uses two independent ACL lookup functions at ingress for IPv4 and IPv6 traffic that can be used for security and, or, QoS-related functions.

When two input ACL filters are applied to a subinterface, the order in which the filters are configured determines the filter processing order. For example, if two input filters F1 and F2 are applied to a subinterface, F1 and F2 are processed sequentially in terms of actions and statistics.

```
--{ +* candidate shared default }--[  ]--
# info with-context acl interface ethernet-1/1.1
    acl {
        interface ethernet-1/1.1 {
            input {
                acl-filter F1 type ipv4 {
                }
                acl-filter F2 type ipv4 {
                }
            }
        }
    }
```

F1 and F2 process matching packets as follows:

- If a packet is accepted by F1, it is then processed by F2, which may accept or drop the packet.
- If a packet is dropped by F1, it is discarded and not processed by F2.
- If the matching entry for F1 and the matching entry for F2 both use the same log, rate-limit, forward, forwarding-class or profile action, then only the action from F1 is used. For the rate-limit policer action, both conforming and discarded packets in F1 are not processed in F2.

### 3.3.6 Attaching an ACL to the management interface

#### Procedure

To control traffic filtering for the management interface, attach the IPv4 or IPv6 ACL for input/output traffic to the subinterface of interface `mgmt0`.

ACLs with actions rate-limit, forward, forwarding-class, or profile cannot be applied to the management interface.

#### Example: Attach an ACL to the mgmt0 interface

```
--{ * candidate shared default }--[  ]--
# acl interface mgmt0.0
--{ * candidate shared default }--[ acl interface mgmt0.0 ]--
# interface-ref interface mgmt0 subinterface 0
--{ * candidate shared default }--[ acl interface mgmt0.0 ]--
# input acl-filter ip_tcp type ipv4
--{ * candidate shared default }--[ acl interface mgmt0.0 input acl-filter ip_tcp type
 ipv4 ]--
# exit
--{ * candidate shared default }--[ acl interface mgmt0.0 input]--
# input acl-filter ipv6_tcp type ipv6
```

#### Example: Verify the configuration

To verify the configuration for the management interface ACL:

```
--{ * candidate shared default }--[  ]--
# info with-context interface mgmt0 acl
    interface mgmt0 {
            acl {
                input {
                    ipv4-filter [
                        ip_tcp
                    ]
                    ipv6-filter [
                        ipv6_tcp
                    ]
                }
            }
        }
    }
```

### 3.3.7 Detaching an ACL from an interface

#### Procedure

To detach an ACL from an interface, enter the `acl interface` context and delete the ACL from the configuration.

#### Example: Detach an ACL from a subinterface

```
--{ * candidate shared default }--[  ]--
# acl interface ethernet-1/16.1
--{ * candidate shared default }--[ acl interface ethernet-1/16.1 ]--
# delete input acl-filter ipv4-filter type ipv4
```

### Example: Verify the configuration

Use the **info acl interface** command to verify that the ACL is no longer part of the subinterface configuration. For example:

```
--{ * candidate shared default }--[  ]--
# info with-context acl interface ethernet-1/16.1
    acl {
        interface ethernet-1/16.1 {
            interface-ref {
                interface ethernet-1/16
                subinterface 1
            }
        }
    }
```

## 3.3.8 Detaching an ACL from the management interface

### Procedure

To detach an ACL from the management interface, enter the `acl mgmt0.0` context and delete the ACL from the configuration.

### Example: Detach an ACL from the management interface

```
--{ * candidate shared default }--[  ]--
# acl interface mgmt0.0
--{ * candidate shared default }--[ acl interface mgmt0.0 ]--
# delete input acl-filter ip_tcp type ipv4
```

### Example: Verify the configuration

To verify that the ACL was detached from the management interface:

```
--{ * candidate shared default }--[  ]--
# info with-context interface mgmt0 acl
    interface mgmt0 {
            acl {
                input {
                    ipv6-filter [
                        ipv6_tcp
                    ]
                }
            }
        }
    }
```

## 3.3.9 Modifying ACLs

### Procedure

You can add entries to an ACL, delete entries from an ACL, and delete the entire ACL from the configuration.

### Example: Add an entry to an ACL

To add an entry to an ACL, enter the context for the ACL, then add the entry. For example, the following commands add an entry to IPv4 ACL `ip_tcp`:

```
--{ * candidate shared default }--[  ]--
# acl acl-filter ip_tcp type ipv4
--{ candidate shared default }--[ acl acl-filter ip_tcp type ipv4 ]--
# entry 65535
--{ candidate shared default }--[ acl acl-filter ip_tcp type ipv4 entry 65535 ]--
# log true
--{ candidate shared default }--[ acl acl-filter ip_tcp type ipv4 entry 65535 ]--
# action drop
```

### Example: Delete an entry in an ACL

To delete an entry in an ACL, use the **delete** command under the context for the ACL and specify the sequence ID of the entry to be deleted. For example, the following commands delete the entry in IPv4 ACL `ip_tcp` with sequence ID 65535:

```
--{ candidate shared default }--[  ]--
# acl acl-filter ip_tcp type ipv4
--{ candidate shared default }--[ acl acl-filter ip_tcp type ipv4 ]--
# delete entry 65535
```

### Example: Delete an entire ACL

To delete the entire ACL, use the **delete** command under the `acl` context. For example, the following commands delete the `ip_tcp` ACL:

```
--{ * candidate shared default }--[  ]--
# acl
--{ candidate shared default }--[ acl ]--
# delete acl-filter ip_tcp type ipv4
```

## 3.3.10 Resequencing ACL entries

### Procedure

To aid in managing complex ACLs that have many entries, you can resequence the ACL entries to set a sequence ID number for the first entry and a constant increment for the sequence ID for subsequent entries.

For example, if you have an ACL with three entries, sequence IDs 123, 124, and 301, you can resequence the entries so that the initial entry has sequence ID 100, and the other two entries have sequence ID 110 and 120.

### Example

The following is an example of an ACL with three entries:

```
--{ candidate shared default }--[ acl ]
# info with-context acl acl-filter ip_tcp type ipv4
    acl {
        acl-filter ip_tcp type ipv4 {
            entry 123 {
                match {
```

```
                    ipv4 {
                        destination-ip {
                            prefix 10.1.2.1/24
                        }
                    }
                }
                action {
                    log true
                    drop {
                    }
                }
            }
            entry 124 {
                match {
                    ipv4 {
                        destination-ip {
                            prefix 10.1.3.1/24
                        }
                    }
                }
                action {
                    log true
                    accept {
                    }
                }
            }
            entry 301 {
                match {
                    ipv4 {
                        destination-ip {
                            prefix 10.1.4.1/24
                        }
                    }
                }
                action {
                    drop {
                    }
                }
            }
        }
    }
```

To resequence the entries in the ACL so that the first entry has sequence ID 100, and the next two entries are incremented by 10, enter the context for the ACL, issue the **tools resequence** command, then specify the initial sequence ID and the increment for the subsequent entries. For example:

```
--{ candidate shared default }--[  ]--
# acl acl-filter ip_tcp type ipv4
--{ candidate shared default }--[ acl acl-filter ip_tcp type ipv4 ]--
# tools resequence start 100 increment 10
```

After you enter the command, the ACL entries are renumbered. For example:

```
--{ candidate shared default }--[ acl ]
# info with-context acl acl-filter ip_tcp type ipv4
    acl {
        acl-filter ip_tcp type ipv4 {
            entry 100 {
                match {
                    ipv4 {
                        destination-ip {
                            prefix 10.1.2.1/24
```

```
                    }
                }
            }
            action {
                log true
                drop {
                }
            }
        }
        entry 110 {
            match {
                ipv4 {
                    destination-ip {
                        prefix 10.1.3.1/24
                    }
                }
            }
            action {
                log true
                accept {
                }
            }
        }
        entry 120 {
            match {
                ipv4 {
                    destination-ip {
                        prefix 10.1.4.1/24
                    }
                }
            }
            action {
                drop {
                }
            }
        }
    }
}
```

## 3.4 Control plane module (CPM) filters

For control plane protection, SR Linux supports ACL policies called CPM filters. You can configure one CPM filter that applies to IPv4 traffic, one that applies to IPv6 traffic, and one that applies to non-IP traffic. When ingress traffic is matched by a CPM filter rule, and it is a terminating packet (that is, it must be extracted to the CPM), then it is processed according to the matching CPM filter rule.

The entries for each CPM filter are installed on every line card. They are evaluated after the input subinterface ACLs and after the packet capture ACLs. CPM filter rules have no effect on locally originating traffic or transit traffic, and they have no interaction with output subinterface ACLs.

When a CPM filter ACL is created, its rules are evaluated against all IPv4 or IPv6 traffic that is locally terminating on the router, regardless of where that traffic entered in terms of network instance, subinterface, linecard, pipeline, and so on. Traffic dropped by CPM filter actions is discarded without ICMP generation.

The default name for the CPM filter `acl-filter` object is `cpm`. SR Linux supports `acl-filter` objects of type `ipv4`, `ipv6`, and `mac` for CPM filtering.

CPM filter ACLs support the following actions:

- accept – Allow the packet through to the next processing function.
- rate-limit policer – Feed the packet to a hardware-based policer, which determines if the packet should be queued by the line card toward the CPM or dropped because a policer rate is exceeded.

  On 7730 SXR platforms, you can configure a rate-limiting policer action in packets per second (pps). For this policer action, you can specify the peak rate pps, maximum-burst size in packets.

  On 7250 IXR platforms, you can configure a rate-limiting policer action in kilobits per second (kbps). For this policer, you can specify a peak rate kbps and maximum burst size in bytes.
- rate-limit system-cpu-policer – If the packet has been extracted to the CPM, feed the packet to a software-based policer, which determines if the packet should be delivered to the CPM application or dropped because a packet-per-second rate is exceeded.
- drop – Discard the packet without ICMP generation.
- log – Send information about the dropped packet to the log application.
- collect-stats – Collect statistics for individual CPM filter entries (7730 SXR systems only)

The system-cpu-policer and policer actions police terminating traffic to ensure that the rate does not exceed a safe limit.

The system-cpu-policer action applies an aggregate rate limit, regardless of ingress line card, while the policer action applies a rate limit to the extracted traffic from each core (7250 IXR) or complex (7220 IXR) associated with the ingress port. You can have both types of policer actions in the same CPM filter entry, or only one of them.

CPM filter rules that apply a system-cpu-policer or policer action do not directly specify the policer parameters; they refer to a generically defined policer. This allows different CPM filter entries, even across multiple ACLs, to use the same policer. Optionally, each policer can be configured as entry-specific, which means a different policer instance is used by each referring filter entry, even if they are part of the same ACL.

### 3.4.1 Creating CPM filters

#### Procedure

To create CPM filters for IPv4 traffic, IPv6 traffic, or non-IP traffic, you create an `acl-filter` of type `ipv4`, `ipv6` or `mac` consisting of entries with match criteria and actions, then assign this `acl-filter` to `.system.control-plane-traffic.input.acl`.

CPM-bound traffic that matches the match criteria is processed according to the specified action. Note that the system includes a default `acl-filter` named `cpm` for CPM filtering.

#### Example: Create an IPv4 CPM filter

The following example creates a CPM filter for IPv4 traffic. IPv4 traffic extracted to the CPM that matches the rule in the filter is processed according to the action configured in the rule. In this example, the matching CPM-bound IPv4 traffic is accepted and rate-limited according to the limit specified by the `sp1` system-cpu-policer.

```
--{ * candidate shared default }--[  ]--
# info with-context acl acl-filter cpm type ipv4
    acl {
        acl-filter cpm type ipv4 {
            entry 1000 {
                match {
```

```
                    ipv4 {
                        protocol tcp
                        destination-ip {
                            prefix 10.1.3.1/32
                        }
                        source-ip {
                            prefix 10.1.5.1/32
                        }
                    }
                    transport {
                        destination-port {
                            value 6789
                        }
                        source-port {
                            value 6722
                        }
                    }
                }
                action {
                    accept {
                        rate-limit {
                            system-cpu-policer sp1
                            policer test_pps_policer
                        }
                    }
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context acl policers system-cpu-policer sp1
    acl {
        policers {
            system-cpu-policer sp1 {
                peak-rate-pps 1000000
                maximum-burst-packet 16
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context acl policers policer test_pps_policer
    acl {
        policers {
            policer test_pps_policer {
                peak-rate-pps 1000
                maximum-burst-packet 64
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context system control-plane-traffic input
    system {
        control-plane-traffic {
            input {
                acl {
                    acl-filter cpm type ipv4 {
                    }
                }
            }
```

```
        }
    }
```

### Example: Create a MAC CPM filter

The following example creates a MAC CPM filter. Non-IP traffic extracted to the CPM that matches the filter rule is processed according to the configured action. In this example, the matching CPM-bound Ethernet frames are accepted and rate-limited according to the limit specified by the `dp1` policer setting.

```
--{ * candidate shared default }--[  ]--
# info with-context acl acl-filter mac01 type mac
    acl {
        acl-filter mac01 type mac {
            entry 100 {
                match {
                    l2 {
                        source-mac {
                            address 00:00:5E:00:53:FF
                            mask 00:00:5E:00:53:00
                        }
                    }
                }
                action {
                    accept {
                        rate-limit {
                            policer dp1
                        }
                    }
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context acl policers policer dp1
    acl {
        policers {
            policer dp1 {
                peak-rate 500000
                max-burst 100000
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context system control-plane-traffic input
    system {
        control-plane-traffic {
            input {
                acl {
                    acl-filter mac01 type mac {
                    }
                }
            }
        }
    }
```

### Example: Filter CPM traffic using an IPv4 prefix list

The following example matches based on the IPv4 prefixes configured in a prefix list. In this example, if the source address of a packet matches one of the prefixes in the list, the packet is dropped.

```
--{ * candidate shared default }--[  ]--
# info with-context acl match-list ipv4-prefix-list pl1
    acl {
        match-list {
            ipv4-prefix-list pl1 {
                description "IPv4 prefix match list"
                prefix 10.10.0.0/16 {
                }
                prefix 10.20.0.0/16 {
                }
                prefix 10.30.0.0/16 {
                }
            }
        }
    }
```

```
--{ +* candidate shared default }--[  ]--
# info with-context acl acl-filter pfilter type ipv4
    acl {
        acl-filter pfilter type ipv4 {
            entry 100 {
                match {
                    ipv4 {
                        source-ip {
                            prefix-list pl1 {
                            }
                        }
                    }
                }
                action {
                    drop {
                    }
                }
            }
        }
    }
```

```
--{ +* candidate shared default }--[  ]--
# info with-context system control-plane-traffic input
    system {
        control-plane-traffic {
            input {
                acl {
                    acl-filter pfilter type ipv4 {
                    }
                }
            }
        }
    }
```

### Example: Match traffic extracted to the CPM from a specific network-instance

On 7730 SXR systems, a CPM filter can use a network-instance as a match condition. The following example creates a CPM filter that matches IGMP protocol traffic in network-instance VRF 100.

```
--{ +* candidate shared default }--[  ]--
```

```
# info with-context acl acl-filter cpm-igmp type ipv4
    acl {
        acl-filter cpm-igmp type ipv4 {
            entry 100 {
                match {
                    network-instance "VRF 100"
                    ipv4 {
                        protocol igmp
                    }
                }
                action {
                    accept {
                    }
                }
            }
        }
    }
```

```
--{ +* candidate shared default }--[  ]--
# info with-context system control-plane-traffic input
    system {
        control-plane-traffic {
            input {
                acl {
                    acl-filter cpm-igmp type ipv4 {
                    }
                }
            }
        }
    }
```

## 3.5 Packet capture filters

For troubleshooting purposes, SR Linux supports ACLs called packet capture filters. When an ingress IP packet on any line card transits through the router, and it matches a rule in a packet capture filter, it is copied and extracted toward the CPM (using the capture-filter extraction queue) and delivered to a Linux virtual Ethernet interface, so that it can be displayed by **tcpdump** (or similar packet capture utility), or encapsulated and forwarded to a remote monitoring system.

Similarly, when an ingress IP packet on any line card terminates locally, and it matches a rule of a packet capture filter, it is extracted toward the CPM (using the normal protocol-based extraction queue), and a header field indicates to the CPM to replicate it (after running the CPM-filter rules) toward the Linux virtual Ethernet interface.

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

The entries for each packet capture filter are installed on every line card. On the line card, the entries are evaluated after the input subinterface ACLs and before the CPM-filter ACLs. On the CPM, the entries in the packet capture filter ACL are evaluated after the CPM-filter entries.

When a packet capture filter ACL is created, its rules are evaluated against all transit and terminating IPv4 or IPv6 traffic that is arriving on any subinterface of the router, regardless of where that traffic entered in terms of network instance, subinterface, linecard, pipeline, and so on. Note that packet capture filter ACL rules cannot override interface filter or system filter ACL drop outcomes; packets dropped by interface filter ACLs or a system filter ACL cannot be mirrored to the control plane.

Each packet capture filter entry has a set of zero or more match conditions, and one of two possible actions: accept and copy. The match conditions are the same as the other filter types. The accept action passes the matching packet to the next stage of processing, without creating a copy. The copy action creates a copy of the matching packet, extracts it toward the CPM and delivers it to the designated virtual Ethernet interface.

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 `acl-filter` of type `ipv4` or `ipv6` named `capture` and specify match conditions and actions. See the "Interactive traffic-monitoring tool" chapter in the *SR Linux Troubleshooting Toolkit Guide* for examples of configuring packet capture filters.

Packet capture filters are not supported on 7730 SXR systems.

## 3.6 System filters

A system filter ACL is an IPv4 or IPv6 ACL that evaluates ingress traffic before all other ACL rules. If an IP packet is dropped by a system filter rule, it is the final disposition of the packet; neither a packet capture ACL copy/accept action, nor an ingress interface ACL accept action, nor a CPM-filter accept action can override the drop action of a system filter.

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

At most one system filter can be defined for IPv4 traffic, and at most one system filter can be defined for IPv6 traffic. System filter ACLs are supported on 7220 IXR-D1/D2/D2L/D3/D3L and 7220 IXR-D4/D5 systems only. They can be applied only at ingress, not egress. System filters are not supported on 7730 SXR systems.

Creating a system filter enables the filter. When a system filter is created, its rules are automatically installed everywhere, meaning they are evaluated against all transit and terminating IPv4 or IPv6 traffic arriving on any subinterface of the router, regardless of where that traffic entered in terms of network instance, subinterface, pipeline, and so on.

A system filter is the only type of filter that can match the outer header of tunneled packets. For VXLAN traffic, this allows you to configure a system filter that matches and drops unauthorized VXLAN tunnel packets before they are decapsulated. The system filter matches the outer header of tunneled packets; they do not filter the payload of VXLAN tunnels.

### 3.6.1 Creating a system filter

#### Procedure

When you apply a system filter, it evaluates all transit and terminating IPv4 arriving on any subinterface of the router. The system filter ACL evaluates the traffic before any other ACL filters.

#### Example

The following is an example of a system filter ACL that filters IPv4 traffic.

```
--{ * candidate shared default }--[  ]--
```

```
# info with-context acl acl-filter system type ipv4
    acl {
        acl-filter system type ipv4 {
            entry 44 {
                match {
                    ipv4 {
                        source-ip {
                            prefix 10.0.0.0/24
                        }
                    }
                }
                action {
                    log true
                    drop {
                    }
                }
            }
        }
    }
```

## 3.7 Rate-limiting action for ACL filters

SR Linux supports a rate-limit action in CPM and interface ACL filters to limit the traffic to a specified rate.

- CPM MAC/IPv4/IPv6 filters support the rate-limit action using both policer and system-cpu-policer templates as described in Control plane module (CPM) filters.
- Interface IPv4/IPv6 filters support the rate-limit action in the input and output directions using policer templates on 7220 IXR-D1/D2/D2L/D3/D3L/D4/D5 platforms.
- Interface IPv4/IPv6 filters support the rate-limit action in the input direction only using policer templates on the 7220 IXR-H4 and 7730 SXR platforms.
- The rate-limit action is not supported for ACL filters assigned to the mgmt0 interface.

### Policer templates

The following table describes the settings available for ACL policer templates and lists the platforms where each setting is supported.

An ACL policer template cannot be configured as an action for an ACL assigned to a mgmt0 subinterface.

An individual ACL policer template can be assigned to interface ACL filters or CPM filters, but not to both filter types.

*Table 13: ACL policer template settings*

| ACL policer template setting | Description | Platform support |
|---|---|---|
| **name** | User-defined name of the ACL policer template. | All |
| **entry-specific** | If set to `true`, a policer instance is created per ACL entry. If set to `false`, the policer instance can be shared between entries of an ACL. | All except 7730 SXR systems<br>On 7730 SXR systems, the **entry-specific** parameter is set to `false` and cannot be configured. |

| ACL policer template setting | Description | Platform support |
|---|---|---|
| **maximum-burst-size** | The maximum burst size in bytes (for kbps policers). | All |
| **maximum-burst-packet** | The maximum burst size in packets (for pps policers). | All |
| **peak-rate-kbps** | The peak information rate in kbps. | All |
| **peak-rate-pps** | The peak information rate in pps. | All<br>(can be applied only to CPM filters) |
| **scope** | Can be `global` or `subinterface`:<br>• `global`: This setting is supported using CPM and interface ACL filters:<br>CPM filters: A single instance of the policer is created per forwarding complex.<br>Interface ACL filters: A single instance of the policer is created per forwarding complex, and per direction, in case the same policer is used in both input and output ACLs.<br>• `subinterface`: This setting is supported using interface ACL filters only. One instance of the policer is created per subinterface and direction the policer is used for, in case the same policer is used in both input and output ACLs. | 7220 IXR-D*x* and 7220 IXR-H*x* systems<br>• On 7220 IXR-D1/D2/D2L/ D3/D3L and 7220 IXR-H4: the global policer cannot be shared between IPv4 and IPv6 ACLs.<br>• On 7220 IXR-D4/D5: the global policer can be shared between IPv4 and IPv6 ACLs.<br>• Policer of scope global requires the ACL setting `subinterface-specific` disabled.<br>• Using 7220 IXR-D1/D2/D2L/ D3/D3L and 7220 IXR-H4: the policer cannot be shared between ACLs of the same subinterface.<br>• Using 7220 IXR-D4/D5: the policer can be shared between ACLs of the same subinterface. |

### System CPU policer templates

The following table describes the settings available for system CPU policer templates and lists the platforms where each setting is supported.

*Table 14: System CPU policer template settings*

| System CPU policer template setting | Description | Platform support |
|---|---|---|
| **name** | User-defined name of the system CPU policer template. | All |
| **entry-specific** | Whether a policer instance is instantiated for each entry or shared between entries.<br>If set to `false`, only one policer instance is created from this template, and it is shared by all entries of all CPM filter ACLs that refer to this policer. | All |
| **maximum-burst-packet** | The maximum depth of the policer bucket in number of packets. | All |
| **peak-rate-pps** | The maximum number of packets per second. | All |

## 3.7.1 Configuring the ACL rate-limiting action

### Procedure

To configure a rate limiting action for an ACL, you configure an ACL policer template and apply it as an action in an ACL entry.

### Example: Configure a rate-limit action for an interface ACL

The following example configures an ACL policer template and specifies it as an action in an IPv4 ACL entry. Traffic that matches this entry on the interface where this ACL is applied is subject to the rate-limiting settings configured in the ACL policer template.

```
--{ * candidate shared default }--[  ]--
# info with-context acl policers policer po1
    acl {
        policers {
            policer po1 {
                peak-rate-kbps 1000000
                maximum-burst-size 500000
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context acl acl-filter a2 type ipv4
    acl {
        acl-filter a2 type ipv4 {
            entry 101 {
```

```
                action {
                    accept {
                        rate-limit {
                            policer po1
                        }
                    }
                }
            }
        }
    }
```

CPU generated traffic is not subject to the egress subinterface rate-limit action; packets still match the entry, but are not subject to the rate-limit.

### Example: Configure a rate-limit action for a CPM filter

The following example configures an ACL policer template and specifies it as an action in a CPM filter entry. The entry-specific parameter is set to true, which creates a policer instance for each ACL entry that specifies this policer template as an action.

```
--{ * candidate shared default }--[  ]--
# info with-context acl policers policer po2
    acl {
        policers {
            policer po2 {
                entry-specific true
                peak-rate-kbps 1000000
                maximum-burst-size 500000
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info with-context system control-plane-traffic input acl acl-filter cpm type ipv4
    acl {
        acl-filter cpm type ipv4 {
            entry 101 {
                action {
                    accept {
                        rate-limit {
                            policer po2
                        }
                    }
                }
            }
        }
    }
```

### Example: Display ACL policer statistics

To display statistics for traffic subject to the ACL policer template, use the **info from state** command.

```
--{ * candidate shared default }--[  ]--
# info with-context from state acl policers policer po2
    acl {
        policers {
            policer po2 {
                peak-rate-kbps 500000
                maximum-burst-size
                statistics {
                    aggregate {
```

```
                        conforming-packets 6
                        conforming-octets 516
                        exceeding-packets 0
                        exceeding-octets 0
                    }
                }
            }
        }
    }
```

## 3.8 Configuring logging for ACLs

You can configure SR Linux to log information about packets that match an ACL entry in the system log.

You can set thresholds for ACL or TCAM resource usage. When utilization of a specified resource reaches the threshold in either the rising or falling direction, it can trigger a log message.

### 3.8.1 Enabling syslog for the ACL subsystem

#### Procedure

If you set the **log** parameter to **true** for the accept or drop action in an ACL entry, information about packets that match the ACL entry is recorded in the system log. You can specify settings for the log file for the ACL subsystem, including the location of the log file, maximum log file size, and the number of log files to keep.

#### Example

The following configuration specifies that the log file for the ACL subsystem be stored in the file `dut1_file`, located in the `/opt/srlinux/bin/logs/srbase` directory. The log file can be a maximum of 1 Mb. When the log file reaches this size, it is renamed using `dut1_file` as its base name. The five most recent log files are kept.

```
--{ candidate shared default }--[  ]--
# system
--{ candidate shared default }--[ system ]--
# info with-context logging file dut1_file
    logging {
        file dut1_file {
            directory /opt/srlinux/bin/logs/srbase/
            rotate 5
            size 1M
            subsystem acl {
            }
        }
    }
```

Ensure that write permission is set for the specified directory path.

#### 3.8.1.1 Syslog entry examples

The following are examples of syslog entries for ACLs.

IPv4 Accept:

```
acl||I Type: Ingress IPv4 Filter: testing Sequence Id: 100 Action: Accept Interface: ethernet-
1/16:1 Packet length: 56 IP Source: 10.1.5.1 Destination: 100.1.3.1 Protocol: 6 TCP Source
 port: 6722 Destination Port: 6789 Flags: SYN
```

IPv4 Drop:

```
acl||I Type: Ingress IPv4 Filter: test Sequence Id: 65535 Action: Drop Interface: ethernet-1/
16:1 Packet length: 44 IP Source: 10.3.2.3 Destination: 100.1.3.1 Protocol: 17 UDP Source port:
 6722 Destination Port: 6789
```

IPv6 Accept:

```
acl||I Type: Ingress IPv6 Filter: tests Sequence Id: 1000 Action: Accept Interface: ethernet-
1/16:1 Packet length: 76 IP Source: 2001:db8:1:5::1 Destination: 2001:10:1:3::1 Protocol: 6 TCP
 Source port: 6722 Destination Port: 6789 Flags: SYN
```

## 3.8.2 Logging ACL resource usage

### Procedure

You can set thresholds for ACL resource usage. When utilization of a specified ACL resource, such as input IPv4 filter instances, reaches the threshold in either the rising or falling direction, it can trigger a log message.

**Note:** This feature is supported on both SXR and IXR platforms.

### Example

The following example sets thresholds for resource usage by input IPv4 filter instances. If the resource usage percentage falls below the `falling-threshold-log` value, a log message of priority `notice` is generated. If the resource usage percentage falls below the `rising-threshold-log` value, a log message of priority `warning` is generated.

```
--{ * candidate shared default }--[  ]--
# info with-context platform resource-monitoring
    platform {
        resource-monitoring {
            acl {
                resource input-ipv4-filter-instances {
                    rising-threshold-log 90
                    falling-threshold-log 90
                }
            }
        }
    }
```

### 3.8.3 Logging TCAM resource usage

#### Procedure

You can set thresholds for Ternary Content Addressable Memory (TCAM) resource usage. When utilization of a specified TCAM resource, such as TCAM used by IPv4 CPM filters, reaches the threshold in either the rising or falling direction, it can trigger a log message.

**Note:** This feature is supported on both SXR and IXR platforms.

#### Example

The following example sets thresholds for TCAM resource usage by IPv4 CPM filters. If the resource usage percentage falls below the `falling-threshold-log` value, a log message of priority `notice` is generated. If the resource usage percentage falls below the `rising-threshold-log` value, a log message of priority `warning` is generated.

```
--{ * candidate shared default }--[  ]--
# info with-context platform resource-monitoring
    platform {
        resource-monitoring {
            tcam {
                resource cpm-capture-ipv4 {
                    rising-threshold-log 90
                    falling-threshold-log 90
                }
            }
        }
    }
```

## 3.9 Collecting and displaying ACL statistics

SR Linux can collect statistics for packets matching an ACL and display statistics for the matched packets. You can display the amount of system resources (TCAM) used by each type of ACL on each line card. ACL statistics can also be displayed using **show** commands.

### 3.9.1 Collecting ACL statistics (7250 IXR, 7220 IXR, and 7215 IXS systems)

#### Procedure

On 7250 IXR, 7220 IXR, and 7215 IXS systems, you can configure an ACL to collect statistics for packets matching the ACL. Statistics can be collected for packets that match each ACL entry, as well as for matching input/output traffic per subinterface.

#### Example: Collect statistics for each ACL entry

The following example configures the ACL to record the number of matching packets for each entry:

```
--{ * candidate shared default }--[  ]--
# info with-context acl acl-filter ip_tcp type ipv4
    acl {
```

```
        acl-filter ip_tcp type ipv4 {
            statistics-per-entry true
        }
    }
```

### Example: Collect per-entry, per-subinterface ACL statistics

By default, if two or more subinterfaces on the same line card reference the same ACL for filtering the same direction of traffic, they use a shared instance of the same ACL in hardware. This means that per-entry statistics (including the number of matched packets and the time stamp of the last matching packet), if enabled, reflect the aggregate of the data gathered for the multiple subinterfaces.

To collect per-entry, per-subinterface statistics, instead of the aggregate of the subinterfaces where the ACL is applied, you can configure an ACL to operate in subinterface-specific mode.

If you change an ACL from subinterface-specific mode to shared mode, or the other way around, during the transition from one mode to the next, traffic continues to be subject to the previous mode until the system resources (TCAM) entries are programmed for the new mode.

The following example configures the ACL to collect statistics for matching packets inbound and outbound on each subinterface:

```
--{ * candidate shared default }--[  ]--
# info acl acl-filter ip_tcp type ipv4
    acl {
        acl-filter ip_tcp type ipv4 {
            subinterface-specific input-and-output
        }
    }
```

You can configure the following values for the **subinterface-specific** parameter:

- **disabled** (the default) – All subinterfaces on a single line card that reference the ACL as an input ACL use a shared filter instance, and all subinterfaces on a single line card that reference the ACL as an output ACL use a shared filter instance.
- **input-only** – All subinterfaces on a single line card that reference the ACL as an output ACL use a shared filter instance, but each subinterface that references the ACL as an input ACL uses its own separate instance of the filter.
- **output-only** – All subinterfaces on a single line card that reference the ACL as an input ACL use a shared filter instance, but each subinterface that references the ACL as an output ACL uses its own separate instance of the filter.
- **input-and-output** – Each subinterface that references the ACL as either an input ACL or an output ACL uses its own separate instance of the filter.

## 3.9.2 Collecting ACL statistics (7730 SXR systems)

### Procedure

On 7730 SXR systems, you can enable statistics collection for individual ACL filter entries by configuring the collect-stats action. The collect-stats action is supported for input and output IPv4/IPv6 interface filters, and records packet and byte match counters for input and output separately. The collect-stats action is also supported for CPM filters.

**Note:** On 7730 SXR systems, you enable the collect-stats action instead of configuring `statistics-per-entry true` setting.

If the same ACL filter is attached to multiple subinterfaces, the statistics collected by the collect-stats action are aggregated for all subinterfaces using the ACL filter.

### Example

The following example configures an ACL entry to record statistics for matching packets:

```
--{ candidate shared default }--[ acl ]--
# info with-context acl acl-filter F1 type ipv4
    acl {
        acl-filter F1 type ipv4 {
            entry 100 {
                match {
                    ipv4 {
                        destination-ip {
                            prefix 10.10.0.0/16
                        }
                    }
                }
                action {
                    collect-stats true
                    accept {
                    }
                }
            }
        }
    }
```

## 3.9.3 Displaying ACL statistics

### Procedure

Use the **info from state** command to display the matched packet statistics and the time of the last match for the interfaces to which the ACL is attached.

### Example

In the following example, the ACL is attached to two interfaces, and statistics are collected for each subinterface:

```
--{ candidate shared default }--[  ]--
# info from state acl acl-filter ip_tcp type ipv4
    subinterface-specific input-and-output
    statistics-per-entry true
    entry 1000 {
        description "Match IP Address TCP Protocol Ports"
        action {
            accept {
                log true
            }
        }
        match {
            destination-address 10.1.3.1/32
            protocol tcp
            source-address 10.1.5.1/32
            destination-port {
```

```
                value 6789
            }
            source-port {
                value 6722
            }
        }
        statistics {
            aggregate {
                in-matched-packets 3000
                in-last-match 2019-07-16T10:53:00.1563Z
                out-matched-packets 0
            }
            per-interface {
                subinterface ethernet-1/16.1 {
                    in-matched-packets 3000
                    in-last-match 2019-07-16T10:53:00.1563Z
                }
            }
        }
    }
    entry 65535 {
        action {
            drop {
                log true
            }
        }
        statistics {
            aggregate {
                in-matched-packets 1000
                in-last-match 2019-07-16T10:53:30.1563Z
            }
            per-interface {
                subinterface ethernet-1/16.1 {
                    in-matched-packets 1000
                    in-last-match 2019-07-16T10:53:30.1563Z
                }
            }
        }
    }
```

If the match criteria changes for an ACL entry, the statistics counter does not reset to zero. To reset the statistics counter for an ACL entry to zero, use the **tools acl clear** command, as described in Clearing ACL statistics.

## 3.9.4 Displaying ACL resource usage

### Procedure

Use the **info from state platform** command to display the amount of system resources (TCAM) used by each type of ACL on each line card.

### Example: Display free-static and free-dynamic resources

The following example shows two different numbers for the remaining (free) TCAM entry resources that are available for input IPv4 ACLs on the line card. The free-static value refers to the available number of resources assuming no additional TCAM banks are dynamically assigned to the ACL type.

The free-dynamic value refers to the available number of resources assuming all unallocated TCAM banks are dedicated to the specified ACL type.

```
--{ candidate shared default }--[  ]--
# info with-context from state platform linecard 1 forwarding-complex 0 tcam resource if-
input-ipv4
    platform {
        linecard 1 {
            forwarding-complex 0 {
                tcam {
                    resource if-input-ipv4 {
                        free-static 2046
                        free-dynamic 18430
                        reserved 2
                        programmed 2
                    }
                }
            }
        }
    }
```

### Example: Display system resources allocated to input IPv4 ACLs

The following example shows the amount of system resources allocated to input IPv4 ACLs on the line card and how much is used and free:

```
--{ candidate shared default }--[  ]--
# info with-context from state platform linecard 1 forwarding-complex 0 acl resource
 input-ipv4-filter-instances
    platform {
        linecard 1 {
            forwarding-complex 0 {
                acl {
                    resource input-ipv4-filter-instances {
                        used 1
                        free 254
                    }
                }
            }
        }
    }
```

## 3.9.5 Clearing ACL statistics

### Procedure

To reset ACL statistics counters to zero, use the **tools acl clear** command. This command can clear statistics at the IPv4 / IPv6 / CPM filter level, ACL entry level, or for an interface or subinterface to which the ACL is attached.

### Example: Clear statistics for an IPv4 filter

```
--{ candidate shared default }--[  ]--
# tools acl acl-filter tcp_ip type ipv4 statistics clear
```

### Example

After this command is executed, the **info from state** output for the IPv4 filter includes a timestamp indicating when the statistics were cleared. For example:

```
--{ candidate shared default }--[  ]--
# info from state acl acl-filter ip_tcp type ipv4
    subinterface-specific output-only
    statistics-per-entry true
    entry 1000 {
        description "Match IP Address TCP Protocol Ports"
        action {
            accept {
                log true
            }
        }
        match {
            destination-address 100.1.3.1/32
            protocol tcp
            source-address 10.1.5.1/32
            destination-port {
                value 6789
            }
            source-port {
                value 6722
            }
        }
        statistics {
            last-clear 2024-03-22T13:53:51.000Z
        }
    }
```

### Example: Clear statistics for a specific IPv4 filter entry

The following example clears statistics for a specific entry in the IPv4 filter:

```
--{ candidate shared default }--[  ]--
# tools acl acl-filter ip_tcp type ipv4 entry 10 statistics clear
```

### Example: Clear statistics for a subinterface for a IPv4 filter entry

The following example clears statistics for a specified subinterface for a specified entry in the IPv4 filter:

```
--{ candidate shared default }--[  ]--
# tools acl acl-filter ip_tcp type ipv4 entry 10 statistics per-interface subinterface 1
 clear
```

## 3.9.6 Displaying ACL statistics using show commands

### Procedure

You can display ACL statistics using relevant **show** commands.

### Example: Display all active ACLs

To display information about all active ACLs, use the **show acl summary** command. For example:

```
--{ candidate shared default }--[  ]--
# show acl summary
-------------------------------------------------------------------------------
Capture Filter ACLs
-------------------------------------------------------------------------------
ipv4-entries: 0
ipv6-entries: 0
-------------------------------------------------------------------------------
IPv4 Filter ACLs
-------------------------------------------------------------------------------
Filter : ip_tcp
Active on : 1 subinterfaces (input) and 0 subinterfaces (output)
Entries : 2
-------------------------------------------------------------------------------
IPv6 Filter ACLs
-------------------------------------------------------------------------------
Filter : ipv6_tcp
Active on : 1 subinterfaces (input) and 0 subinterfaces (output)
Entries : 1
-------------------------------------------------------------------------------
MAC Filter ACLs
-------------------------------------------------------------------------------
Filter   : mac01
Active On: 1 subinterfaces (input) and 0 subinterfaces (output)
Entries  : 5
-------------------------------------------------------------------------------
```

### Example: Display statistics for a specific ACL

You can display statistics for a specific ACL, including how many times each ACL entry was matched on all subinterfaces to which the ACL was applied. For example:

```
--{ candidate shared default }--[  ]--
# show acl acl-filter ip_tcp type ipv4
===============================================================================
Filter        : ip_tcp
SubIf-Specific: disabled
Entry-stats   : no
Entries       : 1
-------------------------------------------------------------------------------
 Subinterface     Input   Output
ethernet-1/16.1   yes     no
-------------------------------------------------------------------------------
Entry 1000
  Match               : protocol=tcp, 10.1.5.1/32(6722-6722)->10.1.3.1/32(6789-6789)
  Action              : accept
  Collect Stats       : false
  Match Packets       : 1000
  Last Match          : 18 seconds ago
  TCAM Entries        : 1 for one subinterface and direction
Entry 65535
  Match               : protocol=<undefined>, any(*)->any(*)
  Action              : drop
  Collect Stats       : false
  Match Packets       : 3000
  Last Match          : 6 minutes ago
  TCAM Entries        : 1 for one subinterface and direction
-------------------------------------------------------------------------------
```

### Example: Display per-interface statistics for each ACL entry

To display per-interface statistics for packets matching each ACL entry, specify the interface name in addition to the ACL name. For example:

```
--{ candidate shared default }--[  ]--
# show acl acl-filter ip_tcp type ipv4 subinterface ethernet-1/16.1
===============================================================================================
Filter        : ip_tcp
SubIf-Specific: disabled
Entry-stats   : no
Entries       : 1
-----------------------------------------------------------------------------------------------
 Subinterface     Input   Output
ethernet-1/16.1   yes     no
-----------------------------------------------------------------------------------------------
Entry 1000
  Match               : protocol=tcp, 10.1.5.1/32(6722-6722)->10.1.3.1/32(6789-6789)
  Action              : accept
  Collect Stats       : false
  Match Packets       : 3000
  Last Match          : 5 minutes ago
  TCAM Entries        : 1 for one subinterface and direction
Entry 65535
  Match               : protocol=<undefined>, any(*)->any(*)
  Action              : drop
  Collect Stats       : false
  Match Packets       : 3000
  Last Match          : 10 minutes ago
  TCAM Entries        : 1 for one subinterface and direction
-----------------------------------------------------------------------------------------------
```

### Example: Display statistics for a CPM filter

To display statistics for packets matching a CPM filter, specify the CPM filter type (IPv4 or IPv6). For example:

```
--{ candidate shared default }--[  ]--
# show acl acl-filter cpm type ipv6
===============================================================================
Filter        : cpm
SubIf-Specific: disabled
Entry-stats   : no
Entries       : 48
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Entry 10
  Match               : next_header=icmp6, any(*)->any(*)
  Action              : accept
  Collect Stats       : false
  Match Packets       : 0
  Last Match          : never
  TCAM Entries        : 1 for one subinterface and direction
Entry 20
  Match               : next_header=icmp6, any(*)->any(*)
  Action              : accept
  Collect Stats       : false
  Match Packets       : 0
  Last Match          : never
  TCAM Entries        : 1 for one subinterface and direction
Entry 1000
  Match               : next_header=<undefined>, any(*)->any(*)
```

```
  Action             : drop
  Collect Stats      : false
  Match Packets      : 0
  Last Match         : never
  TCAM Entries       : 1 for one subinterface and direction
-------------------------------------------------------------------------------
```

# 4 Traffic steering

Traffic steering refers to forwarding traffic in a network-instance based on match conditions and actions defined in a policy or an ACL, as an alternative to forwarding based on entries in a routing table.

- On 7250 IXR and 7220 IXR systems, traffic steering can be configured using policy forwarding. See Traffic steering using policy forwarding.
- On 7730 SXR systems, traffic steering can be configured using ACLs. See Traffic steering using ACLs.
- 7250 IXR Gen 2c+ and 7250 IXR Gen 3 systems support IP tunnel decapsulation using policy forwarding. See Using policy forwarding for tunnel decapsulation.

## 4.1 Traffic steering using policy forwarding

Each forwarding policy is modeled as a sequence of rules, each of which has match conditions and actions. Match conditions specify values for various packet header fields. A packet matches a rule only if all the match conditions evaluate to true. Actions specify the processing to apply to each matching packet.

Each forwarding policy is associated with a specific network-instance. Forwarding policies are not supported for MAC-VRF network-instances. The forwarding policy rules only apply to the ingress subinterfaces of the network-instance. Policy-forwarded packets are classified according to the DSCP policy that is attached to the ingress subinterface.

A forwarding policy can be one of the following types:

- `pbr-policy` – the forwarding policy reflects a policy-based routing (PBR) policy that supports generic PBR actions
- `vrf-selection-policy` – the forwarding policy is used only to classify incoming packets into corresponding network instances. This is the default forwarding policy type.

### Match conditions for forwarding policies

The following table lists the match conditions that can be specified in a forwarding policy:

*Table 15: Match conditions for policy-based forwarding*

| Container | Match condition | Description |
|---|---|---|
| `ipv4` | `protocol` | An IPv4 packet matches this condition if its IP protocol type field matches the specified value. |
| | `dscp-set` | An IPv4 packet matches this condition if its DSCP value matches any of the values in the specified list. |
| | `source-ip.prefix` | An IPv4 packet matches this condition if its source IP address is covered by the specified prefix. |
| | `destination-ip.prefix` | An IPv4 packet matches this condition if its destination IP address is covered by the specified prefix. |

| Container | Match condition | Description |
|---|---|---|
| ipv6 | next-header | An IPv6 packet matches this condition if its first next-header field matches the specified value. |
| | dscp-set | An IPv6 packet matches this condition if its traffic-class value matches any of the values in the specified list. |
| | source-ip.prefix | An IPv6 packet matches this condition if its source IP address is covered by the specified prefix. |
| | destination-ip.prefix | An IPv6 packet matches this condition if its destination IP address is covered by the specified prefix. |
| transport | source-port | An IPv4 packet matches this condition if its transport source port matches the specified port number, name, or port number range.<br>The match condition for ipv4.protocol or ipv6.next-header must be configured as tcp or udp. |
| | destination-port | An IPv4 packet matches this condition if its transport destination port matches the specified port number, name, or port number range.<br>The match condition for ipv4.protocol or ipv6.next-header must be configured as tcp or udp. |

### Actions for forwarding policies

The following table lists the actions you can specify in a forwarding policy and the SR Linux platforms that support each one:

*Table 16: Actions for policy-based forwarding*

| Action | Description | Platform support |
|---|---|---|
| network-instance | Forward matching packets according to IP FIB lookup in the specified network-instance, instead of IP FIB lookup in the network-instance owning the subinterface on which the matching packets arrived. The lookup is done using the IP packet DA.<br>The network-instance specified in the action must be type IP-VRF or default.<br>This action is valid only if the forwarding policy type is vrf-selection-policy. | • Supported on 7220 IXR-D2/D2L/ D3/D3L systems<br>• Not supported on 7220 IXR-D4/D5 systems<br>• Supported on 7250 IXR Gen 2 (IXR-6, IXR-10), 7250 IXR Gen 2c+ (IXR-6e, IXR-10e, IXR-X1b, IXR-X3b), |

| Action | Description | Platform support |
|---|---|---|
| | | and 7250 IXR Gen 3 (IXR-6e, IXR-10e, IXR-18e, IXR-X4) systems<br>• Only supported on routed subinterfaces |
| `next-hop` | Use the specified IP address instead of the DA from the IP header of the packet for the route lookup. The packet is forwarded towards the next-hop that results from this lookup.<br>This action is valid only if the forwarding policy type is `pbr-policy`. | • Supported on 7220 IXR-D2/D2L/ D3/D3L/D4/D5 systems, both on routed and IRB subinterfaces<br>• Supported on 7250 IXR Gen 2 (IXR-6, IXR-10), 7250 IXR Gen 2c+ (IXR-6e, IXR-10e, IXR-X1b, IXR-X3b), and 7250 IXR Gen 3 (IXR-6e, IXR-10e, IXR-18e, IXR-X4) systems, only on routed subinterfaces |
| `network-instance` and `next-hop` | Perform the lookup for matching packets using the `next-hop` IP and in the route table of the specified `network-instance`. The network-instance specified in the action must be type IP-VRF or default.<br>These combined actions are valid only if the forwarding policies is type `pbr-policy`. | • Supported on 7220 IXR-D2/D2L/ D3/D3L/D4/D5 systems, both on routed and IRB subinterfaces<br>• Supported on 7250 IXR Gen 2 (IXR-6, IXR-10), 7250 IXR Gen 2c+ (IXR-6e, IXR-10e, IXR-X1b, IXR-X3b), and 7250 IXR Gen 3 (IXR-6e, IXR-10e, IXR-18e, IXR-X4) systems, only on routed subinterfaces |
| `encapsulate-gre` | Apply GRE encapsulation to matching packets and forward the traffic to configured GRE endpoints.<br>This action is valid only if the policy type is `pbr-policy`. | • Supported on 7250 IXR-X3B/ X1B/6/6e/10/10e/ 18e systems; for 7250 IXR-6e/10e |

| Action | Description | Platform support |
|---|---|---|
| | | Gen 2c+ linecards only |
| `next-hop-group` | Forward matching packets toward the next hops of the referenced next-hop-group, configured under `network-instance.static.next-hop-group`. If the NHG is not programmed, the policy forwarding rule is not programmed either. The NHG must exist in the same network-instance as the forwarding policy. If the NHG has multiple members, the system performs hierarchical ECMP. | • Supported on 7250 IXR Gen 3 (IXR-6e, IXR-10e, IXR-18e, IXR-X4) systems, only on routed subinterfaces |

**Note:** On 7250 IXR Gen 3 devices, a forwarding policy can match IPv4 packets and redirect them to an IPv6 next-hop or to a next-hop-group consisting of IPv6 next hops, as well as match IPv6 packets and redirect them to an IPv4 next-hop or to a next-hop-group made of IPv4 next hops.

### Fallback actions for forwarding policies

If the redirection specified by the forwarding action cannot be performed (such as when the next-hop configured with the `next-hop` action does not resolve) SR Linux can perform a fallback action. SR Linux supports the following fallback actions, which are configured with the **down-override** command:

- `drop` – drop packets that match the forwarding policy rule
- `forward` – forward packets that match the forwarding policy rule according to normal routing lookup
- `skip` – rule is not programmed

SR Linux continuously evaluates whether the redirection can be performed as configured and adapts the programming accordingly.

The following table lists the forwarding actions that support the **down-override** command, as well as their default behavior (that is, when the redirection cannot be performed and **down-override** is not configured):

*Table 17: Forwarding actions that support down-override*

| Action | Default behavior |
|---|---|
| `next-hop` | `skip` |
| `network-instance` and `next-hop` | `skip` |

## 4.1.1 Creating a forwarding policy

### Procedure

To create a forwarding policy, configure the match conditions for the policy and the action to take for packets that meet the match conditions.

### Example: Match based on IPv4 protocol value

The following example configures a forwarding policy that applies to the default network-instance. On subinterfaces where this policy is applied, incoming IPv4 packets that have a value of 4 in their IP protocol field are looked up and forwarded in network-instance red.

```
--{ candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding
    network-instance default {
        policy-forwarding {
            policy 100 {
                description "Sample forwarding Policy"
                rule 1 {
                    action {
                        network-instance red
                    }
                    match {
                        ipv4 {
                            protocol 4
                        }
                    }
                }
            }
        }
    }
```

### Example: Match based on DSCP values

In the following example, incoming packets matching DSCP values 0, 1, or 2 are looked up and forwarded in network-instance blue:

```
--{ * candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding
    network-instance default {
        policy-forwarding {
            policy 101 {
                rule 1 {
                    action {
                        network-instance blue
                    }
                    match {
                        ipv4 {
                            dscp-set [
                                0
                                1
                                2
                            ]
                        }
                    }
                }
            }
        }
    }
```

### Example: Match based on source IP prefix

In the following example, incoming packets whose source IP address matches prefix 10.10.0.0/16 are looked up and forwarded in network-instance green:

```
--{ candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding
```

```
    network-instance default {
        policy-forwarding {
            policy 100 {
                rule 1 {
                    action {
                        network-instance green
                    }
                    match {
                        ipv4 {
                            source-ip {
                                prefix 10.10.0.0/16
                            }
                        }
                    }
                }
            }
        }
    }
```

### Example: Match based on transport source-port

In the following example, packets with TCP source port 179 use 10.10.10.10 for the route lookup. The packets are forwarded to the next-hop that results from this lookup.

```
--{ candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding
    network-instance default {
        policy-forwarding {
            policy p1 {
                type pbr-policy
                rule 1 {
                    action {
                        next-hop 10.10.10.10
                    }
                    match {
                        ipv4 {
                            protocol tcp
                        }
                        transport {
                            source-port 179
                        }
                    }
                }
            }
        }
    }
```

### Example: GRE encapsulation action for matching packets

In the following example, GRE encapsulation is performed on packets that match the policy rule. The matching traffic is redirected and forwarded via GRE encapsulation to the targets specified in the policy action.

```
--{ candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding
    network-instance default {
        policy-forwarding {
            policy 100 {
                type pbr-policy
                rule 1 {
                    action {
                        encapsulate-gre {
```

```
                            target 1 {
                                source 10.10.10.10
                                destination 10.10.1.16/28
                                ip-ttl 12
                            }
                            target 2 {
                                destination 10.10.10.16/28
                            }
                        }
                    }
                    match {
                        ipv4 {
                            protocol tcp
                        }
                    }
                }
            }
        }
    }
```

For each target, you can specify a single destination subnet. The GRE encapsulation distributes matching flows to the component destination addresses that make up the subnet. In this example, the destination subnet for target 1 is 10.10.10.16/28, so flows are distributed across the 16 addresses that make up this subnet.

Traffic that matches the policy is hashed based on the ingress IP header information, which determines which of the configured GRE destination endpoints is used as the destination IP address of the IPv4 GRE header.

Once a packet is encapsulated within the IP-GRE header, it is forwarded to the GRE destination route using the best route within the routing table using available ECMP next-hops, if applicable.

For target 1, a source IP and TTL value are specified, which are applied to the GRE packets originating as a result of this policy action.

### Example: Match based on destination IP prefix

In the following example, incoming packets whose destination IP address matches prefix 10.10.10.10/32 are forwarded via GRE encapsulation to the targets specified in the policy action:

```
--{ candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding
    network-instance default {
        policy-forwarding {
            policy p1 {
                rule 1 {
                    action {
                        encapsulate-gre {
                            target 1 {
                                source 10.20.20.1
                                destination 10.20.1.0/28
                                ip-ttl 4
                            }
                        }
                    }
                    match {
                        ipv4 {
                            protocol udp
                            destination-ip {
                                prefix 10.10.10.10/32
                            }
                        }
```

```
                    }
                }
            }
        }
    }
```

### Example: Perform a fallback action if the next-hop does not resolve

The following example configures the `skip` fallback action, which takes effect if the next-hop specified in the action is not resolvable.

```
--{ +* candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding policy p1
    network-instance default {
        policy-forwarding {
            policy p1 {
                type pbr-policy
                rule 1 {
                    action {
                        next-hop nh1
                        down-override skip
                    }
                    match {
                        ipv4 {
                            protocol tcp
                        }
                    }
                }
                rule 2 {
                    action {
                        next-hop nh2
                    }
                    match {
                        ipv4 {
                            protocol 179
                        }
                    }
                }
            }
        }
    }
```

In this example, two rules are configured, with `action next-hop nh1` configured on rule 1, and `action next-hop nh2` configured on rule 2. With the fallback option set to the default of `skip`, if nh1 is not reachable, rule 1 is skipped, and lookup is performed using rule 2, so traffic is forwarded by nh2.

If the fallback option is set to `forward`, and nh1 is not reachable, then normal routing lookup is performed for matching traffic.

If the fallback option is set to `drop`, and nh1 is not reachable, matching traffic is dropped, and an ICMP unreachable message is sent to the source.

### Example: Forward matching traffic to a next-hop-group

The following example configures a next-hop-group and a forwarding policy that directs IPv4 UDP traffic to the next-hop-group.

```
--{ + candidate shared default }--[  ]--
# info with-context network-instance base static
    network-instance base {
```

```
        static {
            next-hop-group NHG1 {
                next-hop 1 {
                }
            }
            next-hop 1 {
                ip-address 10.10.10.1
                resolve true
            }
        }
    }
```

```
--{ + candidate shared default }--[  ]--
# info with-context network-instance base policy-forwarding policy p1
    network-instance base {
        policy-forwarding {
            policy p1 {
                type pbr-policy
                rule 1 {
                    action {
                        next-hop-group NHG1
                    }
                    match {
                        ipv4 {
                            protocol udp
                        }
                    }
                }
            }
        }
    }
```

### 4.1.2 Applying a forwarding policy

#### Procedure

To activate a forwarding policy, apply the policy to one or more routed subinterfaces of the network-instance configured in the policy.

#### Example

The following example applies a forwarding policy to a subinterface in the default network-instance. The system evaluates ingress packets on the subinterface according to the match conditions in the policy and forwards the matching packets according to the action specified in the policy.

```
--{ candidate shared default }--[  ]--
# info with-context network-instance default policy-forwarding
    network-instance default {
        policy-forwarding {
            interface ethernet-1/1.1 {
                apply-forwarding-policy 100
            }
        }
    }
```

## 4.2 Traffic steering using ACLs

7730 SXR systems support traffic steering using ACLs. The match conditions listed in ACL match conditions can be applied to ACLs used for traffic steering. To configure traffic steering, you configure match conditions and the following ACL actions:

- accept + forward next-hop <address> – Following accept, redirects matching packets to the set of next-hops that result from performing a route lookup in the incoming network-instance using the configured IP address (instead of the IP packet destination address). If the lookup yields no result, or the set of next-hops is down, the packet is dropped and an ICMP destination unreachable message is sent.
- accept + forward next-hop <address> network-instance <name> – Following accept, redirects matching packets to the set of next-hops that result from performing a route lookup in the configured network-instance using the configured IP address (instead of the IP packet destination address). If the lookup yields no result, or the set of next-hops is down, the packet is dropped and an ICMP destination unreachable message is sent.
- accept + forward network-instance <name> – Following accept, redirects matching packets by performing a route lookup in the configured network-instance instead of the incoming network-instance. If the route lookup yields no result or the set of next hops is down, the packet is dropped and an ICMP destination unreachable message is sent.

In all cases the ICMP message is sent from the incoming network-instance.

The following example configures an IPv4 ACL filter entry that causes matching packets to use a specified IP address for the route lookup instead of the DA from the IP header of the packet. The packet is forwarded toward the next-hop that results from this lookup.

```
--{ + candidate shared default }--[  ]--
# info with-context acl acl-filter ts1 type ipv4
    acl {
        acl-filter ts1 type ipv4 {
            entry 100 {
                match {
                    ipv4 {
                        source-ip {
                            prefix 10.10.0.0/16
                        }
                    }
                }
                action {
                    accept {
                        forward {
                            next-hop {
                                address 10.20.20.20
                            }
                        }
                    }
                }
            }
        }
    }
```

## 4.3 Using policy forwarding for tunnel decapsulation

IP tunneling can be used to transport payload packets from point A to point B. This approach adds one or more encapsulation headers at point A, and removes one or more of these encapsulation headers at point B. Between point A and point B, forwarding of the tunneled packets is based on the encapsulation headers and not the original payload headers.

On the node doing the decapsulation (point B), you can configure SR Linux policy forwarding to identify tunneled packets, decide which should be decapsulated, and which headers should be removed. After decapsulation, traffic is forwarded according its inner header destination IP address.

A forwarding policy used for tunnel decapsulation can use an IP prefix or a configured IP prefix-list as a match condition. The following table lists the supported tunnel decapsulation actions.

*Table 18: Tunnel decapsulation actions for policy forwarding*

| Action | Description | Platform support |
|---|---|---|
| `decapsulate-gre` | Remove the Generic Routing Encapsulation (GRE) header from packets matching the rule. | • Supported on 7250 IXR Gen 2c+ (IXR-6e, IXR-10e, IXR-X1b, IXR-X3b), and 7250 IXR Gen 3 (IXR-6e, IXR-10e, IXR-18e, IXR-X4) systems. |
| `decapsulate-gue` | Remove the Generic UDP Encapsulation (GUE) IP-UDP headers from packets matching the rule | • Supported on 7250 IXR Gen 3 (IXR-6e, IXR-10e, IXR-18e, IXR-X4) systems. |

A forwarding policy configured with the `decapsulate-gue` action must also have the **global-decap-policy** option configured. This option applies the forwarding policy to all subinterfaces within the network-instance.

### 4.3.1 Configuring tunnel decapsulation with policy forwarding

#### Procedure

To configure a forwarding policy to decapsulate tunneled packets, specify the match conditions as a prefix or prefix-list and configure the type of tunnel traffic to decapsulate, either GRE or GUE.

#### Example: Decapsulate GRE traffic

The following example configures a forwarding policy to decapsulate GRE traffic to a specific prefix. The forwarding policy removes the GRE headers of matching packets. The policy rules are evaluated for traffic on all IP interfaces in the network-instance.

```
--{ + candidate shared default }--[  ]--
# info with-context network-instance base policy-forwarding
    network-instance base {
        policy-forwarding {
            global-decap-policy p1
            policy p1 {
                type pbr-policy
```

```
                rule 1 {
                    action {
                        decapsulate-gre true
                    }
                    match {
                        ipv4 {
                            destination-ip {
                                prefix 172.16.1.0/24
                            }
                        }
                    }
                }
            }
        }
    }
```

### Example: Decapsulate GUE traffic using a prefix list as match condition

The following example configures a prefix list and uses the prefix list as a match condition in a forwarding policy. The forwarding policy removes the GUE IP-UDP headers of matching packets.

```
--{ + candidate shared default }--[  ]--
# info with-context acl match-list ipv4-prefix-list p_set
    acl {
        match-list {
            ipv4-prefix-list p_set {
                prefix 10.10.10.0/24 {
                }
                prefix 10.10.20.0/24 {
                }
            }
        }
    }
```

```
--{ + candidate shared default }--[  ]--
# info with-context network-instance base policy-forwarding
    network-instance base {
        policy-forwarding {
            global-decap-policy p2
            policy p2 {
                type pbr-policy
                rule 1 {
                    action {
                        decapsulate-gue true
                    }
                    match {
                        ipv4 {
                            protocol udp
                            destination-ip {
                                prefix-list p_set
                            }
                        }
                        transport {
                            destination-port 6080
                        }
                    }
                }
            }
        }
    }
```

# 5 Group-based policy ACLs

Micro-segmentation is a security capability for data centers and enterprises intended to prevent lateral movement of security threats within the network. It divides networks into smaller, isolated zones (known as micro-segments) and establishes rules to restrict traffic between them. A group-based policy (GBP) is a way to control access between network segments using micro-segmentation.

Group-based policies use a type of ACL called a GBP ACL. A GBP ACL is similar to an interface ACL, with the exception that IP addresses are replaced with micro-segmentation group names to enforce security policies within the network-instance.

For information about micro-segmentation and configuring GBPs, see the *SR Linux VPN Services Guide*.

Within the `group-based-policy` context for a network-instance, you can configure a GBP ACL. A GBP ACL contains match criteria specific to GBP filtering, including source and destination group names. The GBP ACL is common for both IPv4 and IPv6 traffic in the network-instance.

The following match conditions are supported concurrently in a GBP ACL:

- source group
- destination group
- TCP/UDP source port (range)
- TCP/UDP destination port (range)
- IP version (IPv4 or IPv6)
- IP protocol type (IPv4 protocol type field or IPv6 next-header field)
- TCP flags, specified as an expression using &, `|`, and `!` logical operators and the TCP flag names: `rst`, `syn`, and `ack`.

The following actions are supported in a GBP ACL:

- accept
- drop
- log
- collect-stats

A GBP ACL filter is not assigned to subinterfaces manually; instead it is configured within the network-instance group-based policy and applies automatically to all ingress packets on all subinterfaces within the network-instance.

Assigning a subinterface IPv4 ACL, IPv6 ACL, or MAC ACL to subinterfaces in a network-instance where a GBP ACL is also configured results in the system executing both ACLs as follows:

- A packet drop by either ACL resulting from a drop or rate-limit action takes priority over the action in the other ACL
- A packet subject to log action in both the GBP ACL entry and subinterface ACL entry is only logged once; in this case, the subinterface ACL takes priority.

### Creating a GBP ACL

To configure a GBP ACL filter within a network-instance, you specify one or more entries consisting of match conditions and the action to take for traffic that matches the conditions.

The following is an example of a GBP ACL filter with one entry. In this example, traffic with source group `grp1` and destination group `grp2` is accepted. The GBP ACL filter applies to both IPv4 and IPv6 traffic.

```
--{ +* candidate shared default }--[  ]--
# info with-context network-instance gbpex group-based-policy acl
    network-instance gbpex {
        group-based-policy {
            acl {
                entry 100 {
                    match {
                        source-group [
                            grp1
                        ]
                        destination-group [
                            grp2
                        ]
                    }
                    action {
                        accept {
                        }
                    }
                }
            }
        }
    }
```

### Displaying GBP ACL information

You can display information about specific GBP ACL entries. For example:

```
--{ running }--[  ]--
# show acl gbp-filter test entry 10
===============================================================================
Network Instance: test
Entries         : 1
-------------------------------------------------------------------------------
Entry 10
  Match               : protocol=<undefined>, [RED:1](*)->[BLUE:2](*)
  Action              : accept
  Collect Stats       : false
  Match Packets       : 0
  Last Match          : never
-------------------------------------------------------------------------------
```

### Clearing GBP ACL statistics

To reset GBP ACL statistics counters to zero, use the **tools network-instance group-based-policy acl statistics clear** command. For example:

```
--{ running }--[  ]--
# tools network-instance gbpex group-based-policy acl statistics clear
```

The following example clears statistics for a specific entry in the GBP ACL:

```
--{ running }--[  ]--
```

```
# tools network-instance gbpex group-based-policy acl entry 10 statistics clear
```

# 6 TCAM allocation on SR Linux devices

Ternary Content Addressable Memory (TCAM) on SR Linux devices can be allocated to system resources either statically or dynamically, based on configuration requirements.

The following sections describe static and dynamic allocation for TCAM banks (known as TCAM slices) and provide details for how SR Linux allocates TCAM on the following devices:

- 7220 IXR-D1
- 7220 IXR-D2 and D3
- 7220 IXR-D4 and D5
- 7220 IXR-H4
- 7250 IXR series

## Static TCAM allocation

Static TCAM refers to TCAM slices that are allocated at initialization and remain constantly reserved for their allocated purpose. Each static TCAM slice has a type, which refers to the type or types of entries that it stores. Examples of different entry types are IPv4 ingress ACL, IPv6 ingress ACL, and so on. A static TCAM slice consumes resources even if has no entries.

When the system is initialized (for example, after a restart) static TCAM slices are allocated first, before the configuration is evaluated and dynamic TCAM slices are created.

## Dynamic TCAM allocation

Dynamic TCAM refers to TCAM slices that are taken from a free or unused pool of TCAM slices and released back to that pool when configuration determines that they are no longer needed.

Each dynamic TCAM slice has a type, which refers to the type or types of entries that it stores. When the configuration of the device has no entries of the type associated with a particular dynamic TCAM slice, there are no allocated slices of that type. The device does not hold onto empty TCAM banks to guarantee a minimum reservation.

The addition of the first entry associated with a particular dynamic TCAM slice attempts to create the required group of slices (slice group). Depending on the entry type (and resulting TCAM key length) the size of the slice group can be one TCAM slice (single-wide TCAM slice type), two TCAM slices (double-wide TCAM slice type), or three TCAM slices (triple-wide TCAM slice type). If a configuration change adds many entries of a new type, multiple slice groups may be needed.

Subsequent configuration changes may add more entries of a type that already has dynamic TCAM slices allocated. When the net number of new entries of that type (that is, additional entries minus deleted entries) exceeds the limit of the current allocation (after filling all empty holes in existing allocated slice groups), the system attempts to allocate new slice groups. There is no artificially imposed limit on the maximum number of slices that can be consumed for an entry type.

If a configuration change requires additional dynamic TCAM slices, and there are not enough free slices to accommodate the new slice groups, even if existing groups are moved to satisfy system constraints on the placement of groups, then the configuration commit is blocked.

If a configuration change requires additional dynamic TCAM slices, and there are enough free slices to accommodate the new slice groups, but only if existing groups are moved to satisfy system constraints on the placement of groups, then the commit is allowed and the TCAM slice relocation operations are initiated automatically. Moving a bank to a new location can take two or three minutes, so an entire operation involving multiple banks could delay the programming of the new entries for up to 20 minutes or more.

The deletion of the last entry associated with a particular dynamic slice group deletes the group and makes the freed slices available to other slice types. If a single configuration commit deletes entries of one type and adds entries of another type, the deleted entries (and any reclamation of slices) are processed first, so that there is a greater chance of successfully allocating the additional slices.

The deletion of some (but not all) entries of a specific type that already has dynamic TCAM slices allocated can leave empty entries in these TCAM slices. This could create a situation where there are N allocated slices for a specific entry type, but considering the empty entries in each of these N slices, fewer than N slices would be needed if the used entries could be moved between slices to consolidate them. This process, called slice compaction, is done automatically during the processing of a configuration transaction that results in a net deletion of entries; it should not be service impacting.

The maximum number of statistics resources that is available to dynamic TCAM entry types remains fixed, and does not scale to the maximum possible size for an entry type. If an ACL rule is configured to have per-entry statistics, but a statistics index resource cannot be allocated, this is indicated by `statistics-resource-allocated = false` in the YANG state for the entry.

### Displaying static and dynamic TCAM usage

Use the **info from state platform** command to display the number of free static and dynamic TCAM entries available to each type of ACL. For example:

```
--{ running }--[  ]--
# info with-context from state platform linecard 1 forwarding-complex 0 tcam resource *
    platform {
        linecard 1 {
            forwarding-complex 0 {
                tcam {
                    resource if-input-ipv4 {
                        free-static 6912
                        free-dynamic 0
                        reserved 0
                        programmed 0
                    }
                    resource if-input-ipv4-qos {
                        free-static 6904
                        free-dynamic 0
                        reserved 8
                        programmed 8
                    }
                    resource if-input-ipv6 {
                        free-static 2304
                        free-dynamic 0
                        reserved 0
                        programmed 0
                    }
                    resource if-input-ipv6-qos {
                        free-static 2262
                        free-dynamic 0
                        reserved 42
                        programmed 42
                    }
                    resource if-input-mac {
                        free-static 2304
```

```
                    free-dynamic 0
                    reserved 0
                    programmed 0
                }
                resource if-input-policer {
                    free-static 1536
                    free-dynamic 0
                    reserved 0
                    programmed 0
                }
                resource if-output-cpm-ipv4 {
                    free-static 1536
                    free-dynamic 0
                    reserved 0
                    programmed 0
                }
                resource if-output-cpm-ipv6 {
                    free-static 512
                    free-dynamic 0
                    reserved 0
                    programmed 0
                }
                resource if-output-cpm-mac {
                    free-static 1536
                    free-dynamic 0
                    reserved 0
                    programmed 0
                }
                resource system-capture-ipv4 {
                    free-static 768
                    free-dynamic 0
                    reserved 0
                    programmed 0
                }
                resource system-capture-ipv6 {
                    free-static 384
                    free-dynamic 0
                    reserved 0
                    programmed 0
                }
            }
        }
    }
}
```

The `free-static` statistic is the number of free entries assuming the number of TCAM slices that are currently allocated to the type of entry remains constant.

The `free-dynamic` statistic is the number of free entries that are possible if all remaining unused TCAM slices are dynamically assigned to this one type of entry.

## 6.1 TCAM allocation on 7220 IXR-D1

On the 7220 IXR-D1, all TCAM slices are allocated statically. The 7220 IXR-D1 does not support dynamic TCAM allocation. The 7220 IXR-D1 has 3 groups of TCAM slices associated with 3 different stages of the forwarding pipeline. Each of these groups is a separate resource pool.

TCAM allocation details for each stage are summarized in the following table.

*Table 19: 7220 IXR-D1 TCAM allocation*

| Stage | TCAM allocation details |
|---|---|
| VFP | There are 4 VFP slices. Each VFP slice provides 512 entries.<br>IPv4 packet capture filter entries require a single slice of single-width entries, providing a maximum of 512 entries per slice. One bank is statically allocated, providing scaling of 512 entries.<br>IPv6 packet capture filter entries require a single slice of double-width entries, providing a maximum of 256 entries per slice. One bank is statically allocated, providing scaling of 256 entries.<br>One bank is unused. |
| IFP | There are 18 IFP slices. Each IFP slice provides 512 entries.<br>Ingress IPv4 filter entries require a single slice of single-width entries, providing a maximum of 512 entries per slice. Two banks are statically allocated, providing scaling of 1024 entries.<br>Ingress IPv6 filter entries require a triple-wide slice, providing a maximum of 512 entries per triple-wide slice. Six banks are statically allocated, providing scaling of 1024 entries.<br>Ingress MAC filter entries require a double-wide slice, providing a maximum of 512 entries per double-wide slice. Zero banks are statically allocated.<br>Six banks are unused. |
| EFP | There are 4 EFP slices. Each EFP slice provides 256 entries.<br>Egress IPv4 and IPv4 CPM filter entries require a single slice of single-width entries, providing a maximum of 256 entries per slice. One bank is statically allocated, providing scaling of 256 entries.<br>Egress IPv6 and IPv6 CPM filter entries require a double-wide slice, providing a maximum of 256 entries per double-wide slice. Two banks are statically allocated, providing scaling of 256 entries.<br>Egress MAC filter entries require a single-wide slice, providing a maximum of 256 entries per slice. Zero banks are statically allocated.<br>One bank is unused. |

## 6.2 TCAM allocation on 7220 IXR-D2/D2L/D3/D3L

The 7220 IXR-D2/D2L/D3/D3L have 3 groups of TCAM slices associated with 3 different stages of the forwarding pipeline. Each of these groups is a separate resource pool. A free slice in one pool is not available to an entry type associated with a different stage of the pipeline. For example, freeing an IFP bank does not provide one more available EFP bank.

The details of each stage in terms of supported dynamic TCAM entry types, total number of TCAM slices, and number of pre-reserved static TCAM slices (with their associated entry types) is summarized in the following table.

*Table 20: 7220 IXR-D2/D3 TCAM allocation*

| Stage | TCAM allocation details |
|---|---|
| VFP | Lookup happens after `MY_STATION` lookup, before tunnel encapsulation (if any) is removed. Used to assign a virtual port (VP) to packets arriving on an untagged bridged subinterface. Also used for capture and system filters.<br>There are 4 VFP slices. Each VFP slice provides 256 entries indexed by a 234-bit key or 128 entries indexed by a 468-bit key (intra-slice double-wide mode). |
| | 1 slice is allocated statically. Entries serve 2 purposes:<br>• to strip the transport VLAN 1 tag from outbound CPU-originated packets (to support egress mirroring of such traffic). This requires 1 entry per system.<br>• to assign a VP to packets arriving on an untagged bridged subinterface. This requires 1 entry per untagged bridged subinterface. |
| | 3 slices are available for dynamic allocation:<br>• Capture IPv4 TCAM entries and system IPv4 TCAM entries can share a slice supporting up to 256 entries; maximum possible scale is 768 entries.<br>• Capture IPv6 TCAM entries and system IPv6 TCAM entries can share a slice supporting up to 128 entries; maximum possible scale is 384 entries.<br>• 256 ingress IPv4 Policy Forwarding entries are supported per bank (intra-slice single-wide mode). There are no restrictions on the placement of intra-slice single-wide slices. Maximum possible ingress IPv4 PF TCAM entries = 3*256<br>• 128 ingress IPv6 Policy Forwarding entries are supported per bank (intra-slice double-wide mode). Maximum possible ingress IPv6 PF TCAM entries = 3*128 |
| IFP | Lookup happens after QoS classification, tunnel decapsulation, and FIB lookup. Used for ingress interface ACLs, ingress subinterface policing, ingress MF QoS classification, VXLAN ES functionality, and CPM extraction (CPU QoS queue assignment).<br>There are 12 IFP slices. Each IFP slice provides 768 entries indexed by a 160-bit key (intra-slice double-wide mode). |
| | 4 slices are allocated statically:<br>• 2 slices for CPU QoS queue assignment<br>• 1 slice for VXLAN ES related functionality<br>• 1 slice for ingress subinterface policing, supporting 1536 entries (intra-slice single-wide mode) |
| | 8 slices are available for dynamic allocation:<br>• 768 ingress IPv4 ACL filter entries are supported per bank (intra-slice double-wide mode). There are no restrictions on the placement of intra-slice double-wide slices. Maximum possible ingress IPv4 TCAM entries = 8*768<br>• 768 ingress IPv6 ACL filter entries are supported by 3 consecutive banks (triple-wide mode). There is a virtual boundary between every group of 3 slices. |

| Stage | TCAM allocation details |
|---|---|
| | The end of a triple-wide group cannot cross any of these virtual boundaries. Maximum possible ingress IPv6 TCAM entries = 2*768<br>• 768 ingress MAC ACL filter entries are supported by 2 consecutive banks (double-wide mode). There is a virtual boundary between every group of 3 slices. The end of a double-wide group cannot cross any of these virtual boundaries but the start of the double-wide group does not have to align with a 3-slice boundary. Maximum possible ingress MAC TCAM entries = 3*768<br>• 768 ingress IPv4 MF QoS classification entries are supported per bank (intra-slice double-wide mode). There are no restrictions on the placement of intra-slice double-wide slices. Maximum possible ingress IPv4 TCAM entries = 8*768<br>• 768 ingress IPv6 MF QoS classification entries are supported by 3 consecutive banks (triple-wide mode). There is a virtual boundary between every group of 3 slices. The end of a triple-wide group cannot cross any of these virtual boundaries. Maximum possible ingress IPv6 TCAM entries = 2*768<br>• 768 ingress IPv4 Policy Forwarding entries are supported per bank (intra-slice double-wide mode). There are no restrictions on the placement of intra-slice double-wide slices. Maximum possible ingress IPv4 PF TCAM entries = 8*768<br>• 768 ingress IPv6 Policy Forwarding entries are supported by 3 consecutive banks (triple-wide mode). There is a virtual boundary between every group of 3 slices. The end of a triple-wide group cannot cross any of these virtual boundaries. Maximum possible ingress IPv6 PF TCAM entries = 2*768 |
| EFP | Lookup happens before final packet modification, after CoS rewrite. Used for out-mirror stats, egress interface ACLs and CPM filter ACLs.<br>There are 4 EFP slices. Each EFP slice provides 512 entries indexed by a 272-bit key. |
| | 1 slice is allocated statically. Entries serve 2 purposes:<br>• ES pruning of local-biased traffic. This requires 1 entry per system.<br>• Egress port mirroring stats. This requires 1 entry per outgoing interface. |
| | 3 slices are available for dynamic allocation:<br>• interface egress IPv4 TCAM entries and CPM-filter IPv4 TCAM entries share a single-wide slice supporting up to 512 entries; maximum possible scale is 1536 entries<br>• interface egress IPv6 TCAM entries and CPM-filter IPv6 TCAM entries share a double-wide slice supporting up to 512 entries; maximum possible scale is 512 entries.<br>IPv6 TCAM entries cannot be added unless all 3 dynamic TCAM banks are free at the time of adding the first IPv6 entry. If there is already an IPv4 or MAC TCAM bank that has been allocated, no IPv6 entries are permitted.<br>• interface egress MAC TCAM entries and CPM-filter MAC TCAM entries share a single-wide slice supporting up to 512 entries; maximum possible scale is 1536 entries |

| Stage | TCAM allocation details |
|---|---|
| | XGS has limitations expanding an EFP slice when the entries have policers; therefore the following restriction is imposed:<br>If a single-wide IPv4 slice has been created, and it has entries with policers (for example. CPM IPv4 filter entries) or entries with a drop and log action, it is not possible to expand the number of IPv4 slices beyond this single slice; conversely, if the number of IPv4 slices was allowed to extend to 2 or more it is not possible to attach a policer or add a drop and log action to any entries in the expanded set of slices |

## 6.3 TCAM allocation on 7220 IXR-D4 and 7220 IXR-D5

The 7220 IXR-D4 and 7220 IXR-D5 have 3 groups of TCAM slices associated with 3 different stages of the forwarding pipeline.

TCAM allocation details for each stage is summarized in the following table.

*Table 21: 7220 IXR-D4 and 7220 IXR-D5 TCAM allocation*

| Stage | TCAM allocation details |
|---|---|
| VFP | There are 4 VFP slices, providing a total of 4*256 = 1024 entries. All slices are allocated statically. |
| | 1 slice:<br>• to strip the transport VLAN 1 tag from outbound-CPU-originated packets (to support egress mirroring of such traffic). This requires 1 entry per system.<br>• to assign a virtual port (VP) to packets arriving on an untagged bridged subinterface. This requires 1 entry per untagged bridged subinterface. |
| | 3 slices for packet capture and system filter entries:<br>• Three physical slices are grouped to support triple-wide entries.<br>• One triple-wide entry is consumed by each IPv4 packet capture filter entry, IPv4 system filter entry, IPv6 packet capture filter entry, or IPv6 system filter entry (the system supports mapping IPv4 groups and IPv6 groups onto the same physical slices); maximum possible scale is 256 entries. |
| IFP | There are 12 IFP slices on 7220 IXR-D4 and 7220 IXR-D5, providing a total of 16K and 24K entries respectively.<br>2 slices are reserved for CPU QoS queue assignment and multicast snooping, providing 2048 entries.<br>1 slice is reserved for VXLAN ES-related functionality providing 1024(7220 IXR-D4)/2048(7220 IXR-D5) entries.<br>The remaining 9 slices are allocated dynamically per application for the following applications (Each slice size is 1024 and 2048 entries for 7220 IXR-D4 and 7220 IXR-D5 respectively): |

| Stage | TCAM allocation details |
|---|---|
| | • MAC ACL: entries require double-wide slices, providing up to 4096(7220 IXR-D4)/8192(7220 IXR-D5) entries<br>• IPv4 ACL: entries require double-wide slices, providing up to 4096(7220 IXR-D4)/8192(7220 IXR-D5) entries<br>• IPv6 ACL: entries require triple-wide slices, providing up to 4096(7220 IXR-D4)/8192(7220 IXR-D5) entries<br>• IPv4 MFC: entries require double-wide slices, providing up to 4096(7220 IXR-D4)/8192(7220 IXR-D5) entries<br>• Policy Based Forwarding: combined IPv4 and IPv6 entries require triple-wide slices, providing up to 4096(7220 IXR-D4)/8192(7220 IXR-D5) entries<br>• Ingress QoS subinterface policing: entries require single slices, providing up to 1024(7220 IXR-D4)/2048(7220 IXR-D5) |
| EFP | There are 4 EFP slices providing a total of 4*512 = 2048 entries |
| | 1 slice is allocated statically. Entries serve 2 purposes:<br>• ES pruning of local-biased traffic. This requires 1 entry per system.<br>• Egress port mirroring statistics. This requires 1 entry per outgoing interface. |
| | 3 slices are available for dynamic allocation<br>• 2 physical slices are grouped to support double-wide entries<br>• 3 physical slices are grouped to support triple-wide entries<br>• 1 triple-wide entry is consumed by each interface egress IPv4 filter entry, interface egress IPv6 filter entry, IPv4 CPM filter entry or IPv6 CPM filter entry (the system supports mapping IPv4 groups and IPv6 groups onto the same physical slices); maximum possible scale is 512 entries<br>• 1 double-wide entry is consumed by each interface egress MAC filter entry or MAC CPM filter entry |

## 6.4 TCAM allocation on 7220 IXR-H4

The 7220 IXR-H4 has 3 groups of TCAM slices associated with 3 different stages of the forwarding pipeline. TCAM allocation details for each stage are summarized in the following table.

*Table 22: 7220 IXR-H4 TCAM allocation*

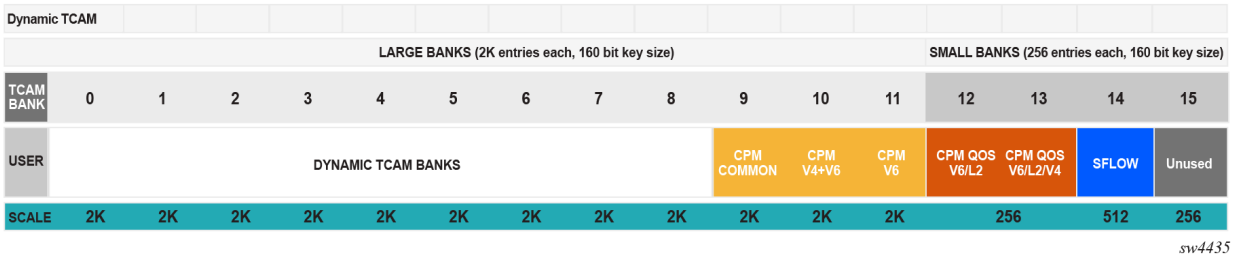
| Stage | TCAM allocation details |
|---|---|
| VFP | There are 4 VFP slices providing a total of 4*256 = 1024 entries |
| IFP | There are 9 IFP slices providing a total of 3072 entries. The first 6 slices (0-5) have 256 entries and the last 3 slices (6-8) have 512 entries. They are allocated statically as follows:<br>• 3 slices (0,1,2) for CPU QoS |

| Stage | TCAM allocation details |
|---|---|
| | 256 CPU QoS entries are supported by group of 3 side-by-side banks (inter-slice triple-wide mode)<br>• 3 slices (3,4,5) for ingress IPv6 ACLs<br>255-3 entries are supported by group of 3 side-by-side banks (inter-slice triple-wide mode)<br>• 2 slices (6,7) for ingress IPv4 ACLs<br>511-3 entries are supported by group of 2 side-by-side banks (inter-slice double-wide mode) |
| EFP | There are 4 EFP slices providing a total of 4*128 = 512 entries. They are allocated statically as follows:<br>• 2 slices for egress + CPM IPv4 ACLs<br>127-11 entries are supported by group of 2 side-by-side banks (inter-slice double-wide mode)<br>• 2 slices for egress + CPM IPv6 ACLs<br>127-11 entries are supported by group of 2 side-by-side banks (inter-slice double-wide mode) |

## 6.5 TCAM allocation on 7250 IXR-6/10/6e/10e

Each forwarding complex on a 7250 IXR-6/10/6e/10e IMM has a TCAM with 12 large banks and 4 small banks. Each of the large banks supports 2K entries each, addressable with a 160-bit key. Each of the small banks supports 256 entries each with a 160-bit key size. This TCAM bank allocation is shown in the following table.

*Figure 1: TCAM allocation on 7250 IXR-6/10/6e/10e platforms*




ACLs can be dynamically allocated to banks 0-8. Requirements for each ACL type are as follows:

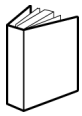
- Ingress IPv4 ACL: entries require single-wide banks that can start at any bank number (and do not need to be contiguous)
- Egress IPv4 ACL: entries require single-wide banks that can start at any bank number (and do not need to be contiguous)
- Ingress IPv6 ACL: entries require double-wide (side-by-side) banks that must start at an even bank number

- Egress IPv6 ACL: entries require double-wide (side-by-side) banks that must start at an even bank number
- Ingress MAC ACL: entries require single-wide banks that can start at any bank number (and do not need to be contiguous)
- Egress MAC ACL: entries require single-wide banks that can start at any bank number (and do not need to be contiguous)
- IPv4 policy forwarding: entries require single-wide banks that can start at any bank number (and do not need to be contiguous), providing up to 9 * 2048 entries
- IPv6 policy forwarding: entries require double-wide (side-by-side) banks that must start at an even bank number, providing up to 4 * 2048 entries

Dynamic TCAM allocation works as follows:

- When a new bank needs to be allocated, the system looks for the first available space, progressing in ascending order from bank 0. If space for a double-wide bank cannot be found, the system attempts to make space by moving the fewest number of single-wide banks.
- When the number of entries required by a particular user drops to a level where a single-wide or double-wide bank can be freed up, the system selects the bank that can create the largest space of free banks, moving entries between banks as necessary.

# Customer document and product support



## Customer documentation

Customer documentation welcome page

## Technical support

Product support portal

## Documentation feedback

Customer documentation feedback