



Implementing Access Lists

- [Understanding Access Lists, on page 1](#)
- [IP Access List Entry Sequence Numbering, on page 6](#)
- [Applying Access Lists, on page 7](#)
- [Understanding Hybrid ACLs, on page 8](#)
- [Configuring IPv4 ACLs, on page 13](#)
- [Configuring IPv6 ACLs, on page 16](#)
- [TCP Flags in ACLs, on page 21](#)
- [Configuring Extended Access Lists, on page 23](#)
- [IPv4 and IPv6 ACL in Class Map, on page 24](#)
- [User-Defined TCAM Keys for IPv4 and IPv6, on page 25](#)
- [Modifying ACLs, on page 29](#)
- [ACLs on Bridge Virtual Interfaces, on page 30](#)
- [Configuring ACL-based Forwarding, on page 32](#)
- [Access Control List Counters, on page 38](#)
- [ACL Statistics Counter, on page 38](#)
- [Configuring ACLs with Fragment Control, on page 39](#)
- [Configuring ACL Filtering by IP Packet Length, on page 43](#)
- [Configuring TTL Matching, on page 46](#)
- [Understanding IP Access List Logging Messages, on page 47](#)
- [Per Interface Statistics, on page 47](#)

Understanding Access Lists

Access lists perform packet filtering to control which packets move through the network and where. Such controls help to limit network traffic and restrict the access of users and devices to the network. Access lists have many uses, and therefore many commands accept a reference to an access list in their command syntax. Access lists can be used to do the following:

An access control list (ACL) consists of one or more access control entries (ACE) that collectively define the network traffic profile. Access control entries (ACE) are entries in an ACL that describe the access rights related to a particular security identifier or user. This profile can then be referenced by Cisco IOS XR software features such as traffic filtering, route filtering, QoS classification, and access control. There are 2 types of ACLs:

- Standard ACLs- Verifies only the source IP address of the packets. Traffic is controlled by the comparison of the address or prefix configured in the ACL, with the source address found in the packet.
- Extended ACLs- Verifies more than just the source address of the packets. Attributes such as destination address, specific IP protocols, User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) port numbers, Differentiated Services Code Point (DSCP), and so on are validated. Traffic is controlled by a comparison of the attributes stated in the ACL with those in the incoming or outgoing packets.

Cisco IOS XR does not differentiate between standard and extended access lists. Standard access list support is provided for backward compatibility.

Purpose of IP Access Lists

- Filter incoming or outgoing packets on an interface.
- Filter packets for mirroring.
- Redirect traffic as required.
- Restrict the contents of routing updates.
- Limit debug output based on an address or protocol.
- Control vty access.
- Identify or classify traffic for advanced features, such as congestion avoidance, congestion management, and priority and custom queueing.

How an IP Access List Works

An access list is a sequential list consisting of permit and deny statements that apply to IP addresses and possibly upper-layer IP protocols. The access list has a name by which it is referenced. Many software commands accept an access list as part of their syntax.

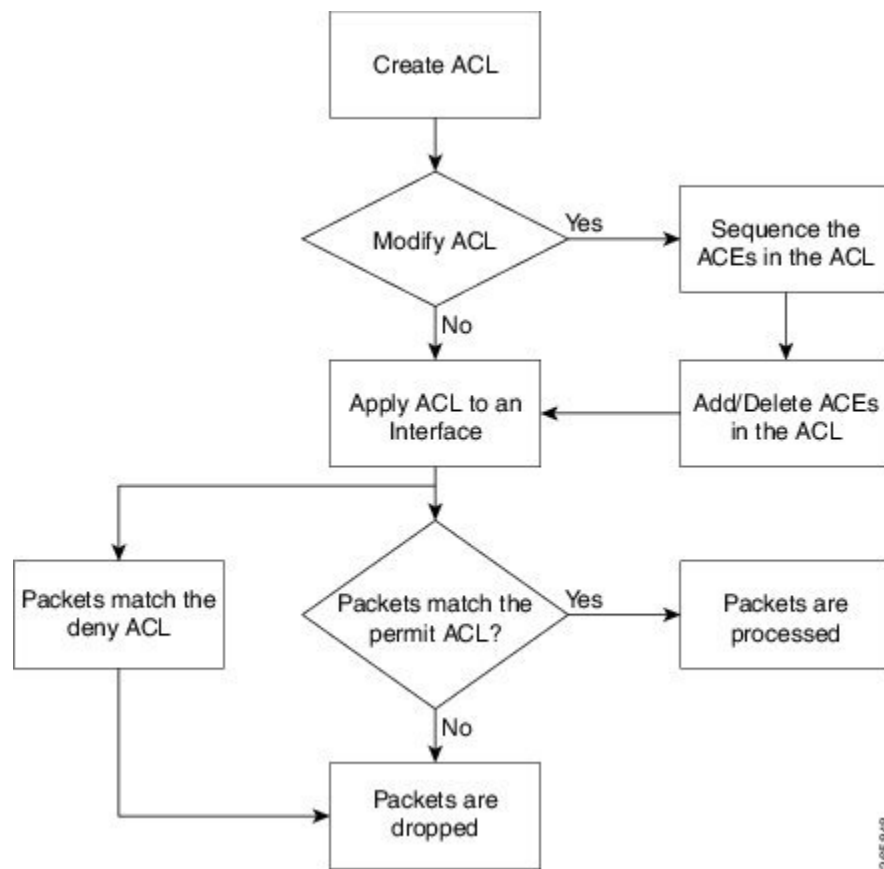
An access list can be configured and named, but it is not in effect until the access list is referenced by a command that accepts an access list. Multiple commands can reference the same access list. An access list can control traffic arriving at the router or leaving the router, but not traffic originating at the router.

Source address and destination addresses are two of the most typical fields in an IP packet on which to base an access list. Specify source addresses to control packets from certain networking devices or hosts. Specify destination addresses to control packets being sent to certain networking devices or hosts.

You can also filter packets on the basis of transport layer information, such as whether the packet is a TCP, UDP, Internet Control Message Protocol (ICMP), or Internet Group Management Protocol (IGMP) packet.

ACL Workflow

The following image illustrates the workflow of an ACL.



IP Access List Process and Rules

Use the following process and rules when configuring an IP access list:

- The software tests the source or destination address or the protocol of each packet being filtered against the conditions in the access list, one condition (permit or deny statement) at a time.
- If a packet does not match an access list statement, the packet is then tested against the next statement in the list.
- If a packet and an access list statement match, the remaining statements in the list are skipped and the packet is permitted or denied as specified in the matched statement. The first entry that the packet matches determines whether the software permits or denies the packet. That is, after the first match, no subsequent entries are considered.
- If the access list denies the address or protocol, the software discards the packet and returns an ICMP Host Unreachable message. ICMP is configurable in the Cisco IOS XR software.
ICMP type and code such as ECHO, ECHO-REPLY, MASK-REPLY, MASK-REQUEST, and so on are supported.
- If no conditions match, the software drops the packet because each access list ends with an unwritten or implicit deny statement. That is, if the packet has not been permitted or denied by the time it was tested against each statement, it is denied.
- The access list should contain at least one permit statement or else all packets are denied.

- Because the software stops testing conditions after the first match, the order of the conditions is critical. The same permit or deny statements specified in a different order could result in a packet being passed under one circumstance and denied in another circumstance.
- Only one access list per interface, per protocol, per direction is allowed.
- Inbound access lists process packets arriving at the router. Incoming packets are processed before being routed to an outbound interface. An inbound access list is efficient because it saves the overhead of routing lookups if the packet is to be discarded because it is denied by the filtering tests. If the packet is permitted by the tests, it is then processed for routing. For inbound lists, permit means continue to process the packet after receiving it on an inbound interface; **deny** means discard the packet.
- Outbound access lists process packets before they leave the router. Incoming packets are routed to the outbound interface and then processed through the outbound access list. For outbound lists, permit means send it to the output buffer; deny means discard the packet.
- An access list can not be removed if that access list is being applied by an access group in use. To remove an access list, remove the access group that is referencing the access list and then remove the access list.
- Before removing an interface, which is configured with an ACL that denies certain traffic, you must remove the ACL and commit your configuration. If this is not done, then some packets are leaked through the interface as soon as the **no interface <interface-name>** command is configured and committed.
- An access list must exist before you can use the **ipv4 | ipv6 access-group** command.

Helpful Hints for Creating IP Access Lists

Consider the following when creating an IP access list:

- Create the access list before applying it to an interface.
- Organize your access list so that more specific references in a network or subnet appear before more general ones.
- To make the purpose of individual statements more easily understood at a glance, you can write a helpful remark before or after any statement.

Source and Destination Addresses

Source address and destination addresses are two of the most typical fields in an IP packet on which to base an access list. Specify source addresses to control packets from certain networking devices or hosts. Specify destination addresses to control packets being sent to certain networking devices or hosts.

ACL Filtering by Wildcard Mask and Implicit Wildcard Mask

Address filtering uses wildcard masking to indicate whether the software checks or ignores corresponding IP address bits when comparing the address bits in an access-list entry to a packet being submitted to the access list. By carefully setting wildcard masks, an administrator can select a single or several IP addresses for permit or deny tests.

Wildcard masking for IP address bits uses the number 1 and the number 0 to specify how the software treats the corresponding IP address bits. A wildcard mask is sometimes referred to as an *inverted mask*, because a 1 and 0 mean the opposite of what they mean in a subnet (network) mask.

- A wildcard mask bit 0 means *check* the corresponding bit value.

- A wildcard mask bit 1 means *ignore* that corresponding bit value.

You do not have to supply a wildcard mask with a source or destination address in an access list statement. If you use the **host** keyword, the software assumes a wildcard mask of 0.0.0.0.

Unlike subnet masks, which require contiguous bits indicating network and subnet to be ones, wildcard masks allow noncontiguous bits in the mask.

You can also use Classless Inter-Domain Routing (CIDR) format (/x) in place of wildcard bits. For example, the IPv4 address 1.2.3.4 0.255.255.255 corresponds to 1.2.3.4/8 and for IPv6 address 2001:db8:abcd:0012:0000:0000:0000:0000 corresponds to 2001:db8:abcd:0012::0/64.

Transport Layer Information

You can filter packets on the basis of transport layer information, such as whether the packet is a TCP, UDP, ICMP, or IGMP packet.

Guidelines and Restrictions for Configuring Access Lists

You must be aware of the following restrictions for configuring access lists.

- IPv4 and IPv6 ACLs are not supported for loopback and interflex interfaces.
- If the Ternary content-addressable memory (TCAM) utilization is high and large ACLs are modified, then an error may occur. During such instances, remove the ACL from the interface and reconfigure the ACL. Later, reapply the ACL to the interface.
- Filtering of Multiprotocol Label Switching (MPLS) packets through interface ACL is not supported.
- Modifying an ACL when it is attached to the interface is supported.
- You can configure an ACL name with a maximum of 64 characters.
- You can configure an ACL name to comprise of only letters and numbers.
- The ACL ID scale per protocol for IPv4 and IPv6 traffic in each direction:
 - The ACL scale for the Cisco 8200 series router is 15 ACL IDs per slice pair, 45 ACL IDs per NPU, 45 ACL IDs per line card, and 45 ACL IDs per router.
 - The ACL scale for the Cisco 8812 series routers with 8800-LC-48H line cards is 15 ACL IDs per slice pair, 30 ACL IDs per NPU, 60 ACL IDs per line card, and 720 ACL IDs per router.
 - The ACL scale for the Cisco 8812 series routers with 8800-LC-36FH-M line cards is 15 ACL IDs per slice pair, 30 ACL IDs per NPU, 120 ACL IDs per line card, and 1440 ACL IDs per router.

Including Comments in Access Lists

You can include comments (remarks) about entries in any named IP access list using the `remark access list` configuration command. The remarks make the access list easier for the network administrator to understand and scan. Each remark line is limited to 255 characters.

The remark can go before or after a **permit** or **deny** statement. You should be consistent about where you put the remark so it is clear which remark describes which **permit** or **deny** statement. For example, it would be confusing to have some remarks before the associated **permit** or **deny** statements and some remarks after the associated statements. Remarks can be sequenced.

Remember to apply the access list to an interface or terminal line after the access list is created.

IP Access List Entry Sequence Numbering

The ability to apply sequence numbers to IP access-list entries simplifies access list changes. Prior to this feature, there was no way to specify the position of an entry within an access list. If a user wanted to insert an entry (statement) in the middle of an existing list, all the entries after the desired position had to be removed, then the new entry was added, and then all the removed entries had to be reentered. This method was cumbersome and error prone.

The IP Access List Entry Sequence Numbering feature allows users to add sequence numbers to access-list entries and resequence them. When you add a new entry, you choose the sequence number so that it is in a desired position in the access list. If necessary, entries currently in the access list can be resequenced to create room to insert the new entry.

Sequence Numbering Behavior

The following details the sequence numbering behavior:

- If entries with no sequence numbers are applied, the first entry is assigned a sequence number of 10, and successive entries are incremented by 10. The maximum configurable sequence number is 2147483643 for IPv4 and IPv6 entries. For other entries, the maximum configurable sequence number is 2147483646. If the generated sequence number exceeds this maximum number, the following message displays:

```
Exceeded maximum sequence number.
```

- If you provide an entry without a sequence number, it is assigned a sequence number that is 10 greater than the last sequence number in that access list and is placed at the end of the list.
- ACL entries can be added without affecting traffic flow and hardware performance.
- If a new access list is entered from global configuration mode, then sequence numbers for that access list are generated automatically.
- Distributed support is provided so that the sequence numbers of entries in the route processor (RP) and line card (LC) are synchronized at all times.
- This feature works with named standard and extended IP access lists. Because the name of an access list can be designated as a number, numbers are acceptable.

Adding Entries with Sequence Numbers: Example

In the following example, a new entry is added to IPv4 access list `acl_5`.

```
ipv4 access-list acl_5
 2 permit ipv4 host 192.0.2.1 any
 5 permit ipv4 host 198.51.100.44 any
10 permit ipv4 host 198.51.100.1 any
20 permit ipv4 host 198.51.100.2 any
configure
ipv4 access-list acl_5
15 permit 203.0.113.1 255.255.255.0
end
```

```
ipv4 access-list acl_5
 2 permit ipv4 host 192.0.2.1 any
 5 permit ipv4 host 198.51.100.44 any
10 permit ipv4 host 198.51.100.1 any
15 permit ipv4 203.0.113.1 255.255.255.0 any
20 permit ipv4 host 198.51.100.2 any
```

Adding Entries Without Sequence Numbers: Example

The following example shows how an entry with no specified sequence number is added to the end of an access list. When an entry is added without a sequence number, it is automatically given a sequence number that puts it at the end of the access list. Because the default increment is 10, the entry will have a sequence number 10 higher than the last entry in the existing access list.

```
configure
ipv4 access-list acl_10
permit 192.0.2.1 255.255.255.0
permit 198.51.100.1 255.255.255.0
permit 203.0.113.1 255.255.255.0
end

ipv4 access-list acl_10
 10 permit ip 192.0.2.1 255.255.255.0 any
 20 permit ip 198.51.100.1 255.255.255.0 any
 30 permit ip 203.0.113.1 255.255.255.0 any

configure
ipv4 access-list acl_10
permit 203.0.113.5 255.255.255.0
end

ipv4 access-list acl_10
 10 permit ip 192.0.2.1 255.255.255.0 any
 20 permit ip 198.51.100.1 255.255.255.0 any
 30 permit ip 203.0.113.1 255.255.255.0 any
 40 permit ip 203.0.113.5 255.255.255.0 any
```

Applying Access Lists

After you create an access list, you must reference the access list to make it work. Access lists can be applied on *either* outbound or inbound interfaces. This section describes guidelines on how to accomplish this task for both terminal lines and network interfaces.

Set identical restrictions on all the virtual terminal lines, because a user can attempt to connect to any of them.

For inbound access lists, after receiving a packet, Cisco IOS XR software checks the source address of the packet against the access list. If the access list permits the address, the software continues to process the packet. If the access list rejects the address, the software discards the packet and returns an ICMP host unreachable message. The ICMP message is configurable.

For outbound access lists, after receiving and routing a packet to a controlled interface, the software checks the source address of the packet against the access list. If the access list permits the address, the software sends the packet. If the access list rejects the address, the software discards the packet and returns an ICMP host unreachable message.

When you apply an access list that has not yet been defined to an interface, the software acts as if the access list has not been applied to the interface and accepts all packets. Note this behavior if you use undefined access lists as a means of security in your network.

Understanding Hybrid ACLs

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
Hybrid ACLs	Release 7.3.1	<p>You can apply compression levels for object-group ACLs and attach up to 4000 ACEs per line card in the ingress direction. This leads to optimal TCAM space usage and resources utilization.</p> <p>The commands modified are:</p> <ul style="list-style-type: none"> • ipv4 access-group • ipv6 access-group

You can use Hybrid ACLs or object-group ACLs to classify users, devices, or protocols into groups so you can have a group-level access control policy. Instead of specifying individual IP addresses, protocols, and port numbers in multiple ACEs, you can specify just the object group in a single ACL.

This feature is very beneficial in large scale networks which currently contain hundreds of ACLs. By using the object-group ACL feature, the number of ACEs per ACL are significantly reduced. Object-group ACLs are also more readable, and easier to manage than conventional ACLs. Using object-group ACLs instead of conventional ACLs optimizes the storage needed in TCAM.

Types of Object-Group ACLs

You can create two types of object-group ACLs on Cisco IOS XR:

- **Network object-group ACLs:** Consist of groups of host IP Addresses and network IP addresses.
- **Port object-group ACLs:** Consist of groups of ports and supporting Layer 3 or Layer 4 protocols.

Compressing ACLs

Object-group ACLs use compression to accommodate the large number of ACEs. Compression is achieved by compressing the following two fields of an ACE:

- Source IP prefix
- Destination IP prefix
- Source port number

There are only two compression levels in the access-group configuration for an ACL on an ingress interface:

- **Compress level 0:** No compression is done on the ACE fields.

In this mode, the object-group ACL behaves like a traditional ACL.

- **Compress level 2:** Two fields (source IP and destination IP) in an ACE are compressed.

In this mode, for ingress traffic, central EM (Exact Match) is used for prefix lookup, and internal TCAM is used for ACE lookup.

Configuring an Object-Group ACL

Before You Begin

You must be aware of the following information that apply to object-group ACLs:

- You can configure ACLs that contain both conventional and object-group ACEs.
- You can modify the objects in an object group dynamically without redefining the object group or the ACE that references the object group.
- You can configure an object-group ACL multiple times with a source group, or a destination group, or both source and destination groups.

Restrictions

Configuring object-group ACLs involves the following restrictions:

- Object-group ACLs can only be configured to an interface. They cannot be used or referenced by applications like SSH, SNMP, NTP.
- To delete an object-group, you must first delete it from all ACLs.
- You cannot configure object-group ACLs along with QoS policies.
- Object-group ACLs are not supported in any policy based configuration.
- Any inline ACE update to an object group ACL clears complete stats of the ACL.

Configuring a Network Object-Group ACL

A network object group can contain a single or multiple network objects.

Restriction

Compression level 2 only supports prefix masks /n, not arbitrary address masks a.b.c.d.

Configuring a Network Object-Group ACL for Compress Level 2

Use the following set of configuration statements to configure a network object-group ACL for an IPv4 address.

```
/* From the global configuration mode, create a network object group. */
Router(config)# object-group network ipv4 netobj1
Router(config-object-group-ipv4)# description my-network-object
Router(config-object-group-ipv4)# host 10.1.1.1
Router(config-object-group-ipv4)# 10.2.1.0 255.255.255.0
Router(config-object-group-ipv4)# range 10.3.1.10 10.3.1.50
```

```

/* Create an access list referencing the object group. */
Router(config)# ipv4 access-list network-object-acl permit ipv4 net-group netobj1 any

/* Apply the access list containing the object group to the desired interface and commit
your configuration. */
Router(config)# interface HundredGigE 0/0/10/3
Router(config-if)# ipv4 address 1.1.1.1/24
Router(config-if)# no shut
Router(config-if)# ipv4 access-group network-object-acl ingress compress level 2
Router(config-if)# commit
Tue Mar 28 10:23:34.106 IST

Router(config-if)# interface HundredGigE 0/0/10/3, changed state to Down
Router(config-if)# interface HundredGigE 0/0/10/3, changed state to Up

Router(config-if)# exit

```

Use the following set of configuration statements to configure a network object-group ACL for an IPv6 address.

```

/* From the global configuration mode, create a network object group. */
Router(config)# object-group network ipv6 netobj1
Router(config-object-group-ipv6)# description my-network-object
Router(config-object-group-ipv6)# host 2001:DB8:1::1
Router(config-object-group-ipv6)# 2001:DB8::1 2001:DB8:0:ABCD::1
Router(config-object-group-ipv6)# range 2001:DB8::2 2001:DB8::5

/* Create an access list referencing the object group. */
Router(config)# ipv6 access-list network-object-acl permit ipv6 net-group netobj1 any

/* Apply the access list containing the object group to the desired interface and commit
your configuration. */
Router(config)# interface HundredGigE 0/0/10/3
Router(config-if)# ipv6 address 2001:DB8::1/32
Router(config-if)# no shut
Router(config-if)# ipv6 access-group network-object-acl ingress compress level 2
Router(config-if)# commit
Tue Mar 28 10:23:34.106 IST

Router(config)# interface HundredGigE 0/0/10/3, changed state to Down
Router(config)# interface HundredGigE 0/0/10/3, changed state to Up

Router(config-if)# exit

```

Running Configuration

Validate your configuration by using the **show run** command.

```

Router(config)# show run
Tue Mar 28 10:37:55.737 IST

Building configuration...
!! IOS XR Configuration 0.0.0
...

!
object-group network ipv4 netobj1
 10.2.1.0/24
 host 10.1.1.1
 range 10.3.1.10 10.3.1.50
 description my-network-object

```

```

!
!
ipv4 access-list network-object-acl
  10 permit ipv4 net-group netobj1 any
!
interface HundredGigE 0/0/10/3
  ipv4 address 1.1.1.1 255.255.255.0
  ipv4 access-group network-object-acl ingress compress level 2
!

```

You have successfully configured a network object-group ACL for compression level 2.

Configuring a Port Object-Group ACL

A port object-group can contain a single or multiple port objects.

Restriction

If you configure port object-group ACLs with compression level 2, it does not result in a lower number of TCAM entries because the compression algorithm only uses the source and destination prefixes and not the port or protocol numbers.

Configuration

Use the following set of configuration statements to configure a port object-group ACL.

```

/* From the global configuration mode, create a port object group, and commit your
configuration. */
Router(config)# object-group port portobj1
Router(config-object-group-ipv4)# description my-port-object
Router(config-object-group-ipv4)# eq bgp
Router(config-object-group-ipv4)# range 100 200
Router(config-object-group-ipv4)# commit
Router(config-object-group-ipv4)# exit

/* Create an access list referencing the object group. */
Router(config)# ipv4 access-list port-object-acl permit ipv4 net-group portobj1

/* Apply the access list containing the object group to the desired interface and commit
your configuration. */
Router(config)# interface HundredGigE 0/0/10/3
Router(config-if)# ipv4 address 2.2.2.2/24
Router(config-if)# ipv4 access-group port-obj-acl ingress compress level 2
Router(config-if)# no shut
Router(config-if)# commit
Tue Mar 28 10:23:34.106 IST

Router(config-if)# interface HundredGigE 0/0/10/3, changed state to Down
Router(config-if)# interface HundredGigE 0/0/10/3, changed state to Up

Router(config-if)# (config-if)# exit

```

Running Configuration

Confirm your configuration.

```

Router(config-if)# (config)# show run
Tue Mar 28 10:37:55.737 IST

```

```

Building configuration...
!! IOS XR Configuration 0.0.0
...
object-group port portobj1
  eq bgp
  range 100 200
!

ipv4 access-list port-object-acl
  10 permit tcp net-group portobj1
!
interface HundredGigE 0/0/10/3
  ipv4 access-group port-obj-acl ingress compress level 2
!
end
!

```

You have successfully configured a port object-group ACL.

Verifying Object-Group ACL Compression

You can use the commands described in this section to verify the configured object-group ACLs in operation and the compression of the ACEs in the ACL.



Note The outputs provided in this section are a standalone sample and are not related to the configurations provided in the preceding sections.

Verification

Use the following set of verification commands to verify object-group ACL compression.

```

/* Verify the entries of the ACL in operation. */

Router# show access-lists ipv4 network-object-acl hardware ingress location 0/0/CPU0
ipv4 access-list network-object-acl
40 permit ospf net-group n_192.168.0.0_16 any (20898463272 matches)
70 permit tcp any net-group CORP_ALL_V4 established
100 permit udp net-group INTERNAL port-group KERBEROS_UDP net-group CORP_ALL_V4
130 permit udp net-group INTERNAL port-group DNS_UDP net-group CORP_ALL_V4
160 permit udp net-group INTERNAL port-group NTP net-group CORP_ALL_V4
190 permit udp net-group INTERNAL port-group LDAP_UDP net-group CORP_ALL_V4
...
1500 permit udp net-group VLAN60_SECURITY net-group h_192.168.77.242 port-group
UDP_50000-50100
1530 deny ipv4 net-group VLAN60_SECURITY any log (20891956640 matches)
...

/* Verify the ACE compression in the ACL. */
Router# show access-lists ipv4 network-object-acl hardware ingress verify location 0/0/CPU0
Verifying TCAM entries for network-object-acl
Please wait...

      INTF      NPU lookup  ACL # intf Total  compression Total  result failed(Entry) TCAM
entries

```

type	ID	shared	ACES	prefix-type	Entries	ACE SEQ #	verified

HundredGigE 0_0_10_3 (ifhandle: 0x1c8)							
1	IPV4	2	1	247	COMPRESSED	810	passed
810					SRC IP	2746	passed
2746					DEST IP	3413	passed
3413					SRC PORT	340	passed
340							

You have successfully verified the compression of ACEs within an ACL.



Note The command `show access-lists access-list-name hardware ingress detail location location` displays compressed output for source and destination IP addresses when the `detail` keyword is used while attaching ACLs to interfaces.

Configuring IPv4 ACLs

This section describes the basic configuration of IPv4 ingress and egress ACLs.

Notes and Restrictions for Configuring IPv4 Ingress ACLs

IPv4 ingress ACLs are characterized by the following behavior for Cisco 8000 Series Routers. These restrictions are subject to change with respect to other platforms.

- Ingress IPv4 ACLs are supported on all interfaces except management interfaces.
- In Fixed system, maximum number of ACLs allowed per NPU is 45, In distributed system, maximum number of ACLs allowed per NPU is 30.
- Packet Length is not supported.
- ACL logging with input interface (using the **log-input** keyword) is not supported.

Notes and Restrictions for Configuring IPv4 Egress ACLs

IPv4 egress ACLs are characterized by the following behavior.

- ACL is not supported on Management interface on egress direction.
- ACL logging is not supported on egress direction.

Configuring an Ingress IPv4 ACL on a HundredGigE Interface

Use the following configuration to configure an ingress IPv4 ACL on a HundredGigE interface.

```
/* Configure a HundredGigE interface with an IPv4 address */
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# ipv4 address 192.0.2.1 255.255.255.0
```

```

Router(config-if)# no shut
Router(config-if)# commit
Thu Jul 11 08:46:51.930 UTC
Router(config-if)# exit

/* Verify if the interface is up */
Router(config)# do show ipv4 interface brief

Thu Jul 11 08:46:51.930 UTC
Interface                IP-Address      Status          Protocol Vrf-Name
HundredGigE 0/0/0/0        192.0.2.1      Up              Up       default

/* Configure an IPv4 ingress ACL */
Router(config)# ipv4 access-list V4-ACL-INGRESS
Router(config-ipv4-acl)# 10 permit tcp 192.0.2.2 255.255.255.0 any
Router(config-ipv4-acl)# 20 deny udp any any
Router(config-ipv4-acl)# 30 permit ipv4 192.0.2.64 255.255.255.0 any
Router(config-ipv4-acl)# commit
Thu Jul 11 08:55:12.806 UTC

/* Verify the ingress ACL creation */
Router(config)# do show access-lists ipv4
Thu Jul 11 08:55:44.824 UTC
...
ipv4 access-list V4-ACL-INGRESS
  10 permit tcp 192.0.2.2 255.255.255.0 any
  20 deny udp any any
  30 permit ipv4 192.0.2.64 255.255.255.0 any

/* Apply the ingress ACL to the HundredGigE interface */
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# ipv4 access-group V4-ACL-INGRESS ingress
Router(config-if)# commit
Thu Jul 11 09:01:26.744 UTC
Router(config-if)# exit

/* Verify if the ingress ACL has been successfully applied to the interface */
Router(config)# do show ipv4 interface
Thu Jul 11 09:01:50.445 UTC
HundredGigE 0/0/0/0 is Up, ipv4 protocol is Up
  Vrf is default (vrfid 0x60000000)
  Internet address is 192.0.2.1/24
  MTU is 1514 (1500 is available to IP)
  Helper address is not set
  Directed broadcast forwarding is disabled
  Outgoing access list is not set
  Inbound common access list is not set, access list is V4-ACL-INGRESS
  Proxy ARP is disabled
  ICMP redirects are never sent
  ICMP unreachable are always sent
  ICMP mask replies are never sent
  Table Id is 0xe0000000

```

You have successfully configured an IPv4 ingress ACL on a HundredGigE interface.

Configuring an Egress IPv4 ACL on a HundredGigE Interface

Use the following configuration to configure an egress IPv4 ACL on a HundredGigE interface.

```

/* Configure a HundredGigE interface with an IPv4 address */
Router(config)# interface HundredGigE 0/0/0/0

```

```

Router(config-if)# ipv4 address 198.51.100.1 255.255.255.0
Router(config-if)# no shut
Router(config-if)# commit
Thu Jul 11 08:55:12.806 UTC
Router(config-if)# exit

/* Verify if the interface is up */
Router(config)# do show ipv4 interface brief
Thu Jul 11 08:55:44.824 UTC

Interface                IP-Address      Status          Protocol Vrf-Name
HundredGigE 0/0/0/0      192.0.2.1       Up             Up       default
HundredGigE 0/0/0/1      198.51.100.1    Up             Up       default

/* Configure an IPv4 egress ACL */
Router(config)# ipv4 access-list V4-ACL-EGRESS
Router(config-ipv4-acl)# 10 permit ipv4 203.0.113.1 255.255.255.0 192.0.2.1 0.255.255.255
Router(config-ipv4-acl)# 20 deny ipv4 any any
Router(config-ipv4-acl)# commit
Thu Jul 11 08:59:10.093 UTC

/* Verify the egress ACL creation */
Router(config)# do show access-lists ipv4
Thu Jan 25 10:25:19.896 IST
ipv4 access-list V4-ACL-EGRESS
10 permit ipv4 203.0.113.1 255.255.255.0 192.0.2.1 255.255.255.0
20 deny ipv4 any any
...

/* Apply the egress ACL to the HundredGigE interface */
Router(config)# interface HundredGigE 0/0/0/1
Router(config-if)# ipv4 access-group V4-ACL-EGRESS egress
Router(config-if)# commit
Thu Jul 11 09:19:49.569 UTC
Router(config-if)# exit

/* Verify if the egress ACL has been successfully applied to the interface */
Router(config)# do show ipv4 interface
Thu Jul 11 09:01:50.445 UTC
HundredGigE 0/0/0/1 is Up, ipv4 protocol is Up
  Vrf is default (vrfid 0x60000000)
  Internet address is 198.51.100/24
  MTU is 1514 (1500 is available to IP)
  Helper address is not set
  Directed broadcast forwarding is disabled
  Outgoing access list is V4-ACL-EGRESS
  Inbound common access list is not set, access list is not set
  Proxy ARP is disabled
  ICMP redirects are never sent
  ICMP unreachable are always sent
  ICMP mask replies are never sent
  Table Id is 0xe0000000
...

```

You have successfully configured an IPv4 egress ACL on a HundredGigE interface. For more information on logging messages, see [Understanding IP Access List Logging Messages, on page 47](#).

Configuring IPv6 ACLs

You can filter IP version 6 (IPv6) traffic by creating IPv6 access control lists (ACLs) and applying them to interfaces similar to the way that you create and apply IP version 4 (IPv4) named ACLs.

Restrictions and Guidelines

The following restrictions and guidelines apply while configuring IPv6 ACLs:

- Ingress IPv6 ACLs are supported on all interfaces.
- From Release 7.3.1 onwards, the maximum number of ACLs allowed per router is 126.
In earlier releases, for the Cisco 8100 and 8200 Series fixed chassis, the maximum number of ACLs allowed per router is 45. For the Cisco 8800 modular chassis, the maximum number of ACLs allowed per router is 30.
- ACL logging with input interface (using the **log-input** keyword) is not supported.
- Packet Length (using the **pkt-length** keyword) is not supported.
- In IPv6 Egress ACLs, TCP flag filtering does not function for IPv6 packets with a fragmentation header. As a result, IPv6 packets with both a fragmentation header and a TCP header (ACK+SYN flags) are not appropriately filtered by the ACL rules.

Configuring an Ingress IPv6 ACL on a HundredGigE Interface

Use the following configuration to configure an ingress IPv6 ACL on a HundredGigE interface.

```
/* Configure a HundredGigE interface with an IPv6 address */
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# ipv6 address 2001::1/64
Router(config-if)# no shut
Router(config-if)# commit
Thu Jul 11 09:28:07.759 UTC
Router(config-if)# exit

/* Verify if the interface is up */
Router(config)# do show ipv6 interface brief
Thu Jul 11 09:28:43.657 UTC
HundredGigE 0/0/0/0 [Up/Up]
    fe80::bd:b9ff:fea9:5606
    2001::1
...

/* Configure an IPv6 ingress ACL */
Router(config)# ipv6 access-list V6-INGRESS-ACL
Router(config-ipv6-acl)# 10 permit ipv6 any any
Router(config-ipv6-acl)# 20 deny udp any any
Router(config-ipv6-acl)# commit
Thu Jul 11 09:41:02.625 UTC
Router(config-ipv6-acl)# exit

/* Verify the ingress ACL creation */
Router(config)# do show access-lists ipv6
Thu Jul 11 09:41:37.260 UTC
ipv6 access-list V6-INGRESS-ACL
```



```

10 permit ipv6 any any
20 deny udp any any

```

```

/* Apply the ingress ACL to the HundredGigE interface */
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# ipv6 access-group V6-INGRESS-ACL ingress
Router(config-if)# commit
Thu Jul 11 09:43:59.733 UTC
Router(config-if)# exit

/* Verify if the ingress ACL has been successfully applied to the interface */
Router(config)# do show ipv6 interface
Thu Jan 25 11:34:08.028 IST
HundredGigE 0/0/0/0 is Up, ipv6 protocol is Up, Vrfid is default (0x60000000)
  IPv6 is enabled, link-local address is fe80::bd:b9ff:fea9:5606
  Global unicast address(es):
    2001::1, subnet is 1001::/64
  Joined group address(es): ff02::1:ff00:1 ff02::1:ffa9:5606 ff02::2
    ff02::1
  MTU is 1514 (1500 is available to IPv6)
  ICMP redirects are disabled
  ICMP unreachable are enabled
  ND DAD is enabled, number of DAD attempts 1
  ND reachable time is 0 milliseconds
  ND cache entry limit is 1000000000
  ND advertised retransmit interval is 0 milliseconds
  Hosts use stateless autoconfig for addresses.
  Outgoing access list is not set
  Inbound common access list is not set, access list is V6-INGRESS-ACL
  Table Id is 0xe0800000
  Complete protocol adjacency: 0
  Complete glean adjacency: 0
  Incomplete protocol adjacency: 0
  Incomplete glean adjacency: 0
  Dropped protocol request: 0
  Dropped glean request: 0
...

```

You have successfully configured an IPv6 ingress ACL on a HundredGigE interface.

Configuring an Egress IPv6 ACL on a HundredGigE Interface

Use the following configuration steps to configure an egress IPv6 ACL on a HundredGigE interface.

```

/* Configure a HundredGigE interface with an IPv6 address */
Router(config)# interface HundredGigE 0/0/0/1
Router(config-if)# ipv6 address 2001::1/64
Router(config-if)# no shut
Router(config-if)# commit
Thu Jan 25 11:41:25.778 IST
Router(config-if)# exit

/* Verify if the interface is up */
Router(config)# do show ipv6 interface brief
Thu Jul 11 09:47:50.812 UTC
HundredGigE 0/0/0/0 [Up/Up]
  fe80::bd:b9ff:fea9:5606
  1001::1
HundredGigE 0/0/0/1 [Up/Up]
  fe80::23:e9ff:fea8:a44e
  2001::1

```

```

/* Configure an IPv6 egress ACL */
Router(config)# ipv6 access-list V6-EGRESS-ACL
Router(config-ipv6-acl)# 10 permit ipv6 any any
Router(config-ipv6-acl)# 20 deny udp any any
Router(config-ipv6-acl)# commit
Thu Jul 11 09:50:40.566 UTC
Router(config-ipv6-acl)# exit

/* Verify the egress ACL creation */
Router(config)# do show access-lists ipv6
Thu Jul 11 09:51:16.687 UTC
ipv6 access-list V6-EGRESS-ACL
  10 permit ipv6 any any
  20 deny udp any any
...

/* Apply the egress ACL to the HundredGigE interface */
Router(config)# interface HundredGigE 0/0/0/1
Router(config-if)# ipv6 access-group V6-EGRESS-ACL egress
Router(config-if)# commit
Thu Jul 11 09:52:57.751 UTC
Router(config-if)# exit

/* Verify if the egress ACL has been successfully applied to the interface */
Router(config)# do show ipv6 interface
Thu Jul 11 09:53:41.365 UTC
...
HundredGigE is Up, ipv6 protocol is Up, Vrfid is default (0x60000000)
  IPv6 is enabled, link-local address is fe80::23:e9ff:fea8:a44e
  Global unicast address(es):
    2001::1, subnet is 2001::/64
  Joined group address(es): ff02::1:ff00:1 ff02::1:ffa8:a44e ff02::2
    ff02::1
  MTU is 1514 (1500 is available to IPv6)
  ICMP redirects are disabled
  ICMP unreachable are enabled
  ND DAD is enabled, number of DAD attempts 1
  ND reachable time is 0 milliseconds
  ND cache entry limit is 1000000000
  ND advertised retransmit interval is 0 milliseconds
  Hosts use stateless autoconfig for addresses.
Outgoing access list is V6-EGRESS-ACL
  Inbound common access list is not set, access list is not set
  Table Id is 0xe0800000
  Complete protocol adjacency: 0
  Complete glean adjacency: 0
  Incomplete protocol adjacency: 0
  Incomplete glean adjacency: 0
  Dropped protocol request: 0
  Dropped glean request: 0
...

```

You have successfully configured an IPv6 egress ACL on a HundredGigE interface.

Configuring Ingress and Egress IPv6 ACLs on Bundle Interfaces

Use the following configuration to configure ingress and egress IPv6 ACLs on a bundle interface.

```

/* Configure a bundle interface with an IPv6 address */
Router(config)# interface Bundle-Ether 1
Router(config-if)# ipv6 address 2001::1/64
Router(config-if)# no shut

```

```

Router(config-if)# commit
Thu Jul 11 09:56:40.603 UTC
Router(config-if)# exit

/* Configure an IPv6 egress ACL */
Router(config)# ipv6 access-list V6-EGRESS-ACL-bundle interface
Router(config-ipv6-acl)# 10 permit tcp any any range 3000 4000
Router(config-ipv6-acl)# 20 permit ipv6 any any
Router(config-ipv6-acl)# commit
Thu Jul 11 10:02:34.568 UTC
Router(config-ipv6-acl)# exit

/* Configure an IPv6 ingress ACL to deny ingress traffic on the bundle interface */
Router(config)# ipv6 access-list V6-DENY-INGRESS-ACL
Router(config-ipv6-acl)# 10 deny ipv6 any any
Router(config-ipv6-acl)# commit
Thu Jul 11 10:03:43.411 UTC
Router(config-ipv6-acl)# exit

/* Verify the egress and ingress ACL creation */
Router(config)# do show access-lists ipv6
Thu Jul 11 10:04:35.798 UTC
ipv6 access-list V6-DENY-INGRESS-ACL
  10 deny ipv6 any any
ipv6 access-list V6-EGRESS-ACL-BI
  10 permit tcp any any range 3000 4000
  20 permit ipv6 any any
...

/* Apply the egress and ingress ACLs to the bundle interface */
Router(config)# interface Bundle-Ether 1
Router(config-if)# ipv6 access-group V6-EGRESS-ACL-BI egress
Router(config-if)# ipv6 access-group V6-DENY-INGRESS-ACL ingress
Router(config-if)# commit
Thu Jul 11 10:06:06.452 UTC
Router(config-if)# exit

/* Verify if the ACLs have been successfully applied to the interface */
Router(config)# do show ipv6 interface
Thu Jul 11 10:06:49.975 UTC
...
Bundle-Ether1 is Down, ipv6 protocol is Down, Vrfid is default (0x60000000)
IPv6 is enabled, link-local address is fe80::1:10ff:fe87:8d04 [TENTATIVE]
Global unicast address(es):
  2001::1, subnet is 2001::/64 [TENTATIVE]
  Joined group address(es): ff02::2 ff02::1
  MTU is 1514 (1500 is available to IPv6)
  ICMP redirects are disabled
  ICMP unreachable are enabled
  ND DAD is enabled, number of DAD attempts 1
  ND reachable time is 0 milliseconds
  ND cache entry limit is 1000000000
  ND advertised retransmit interval is 0 milliseconds
  ND router advertisements are sent every 160 to 240 seconds
  ND router advertisements live for 1800 seconds
  Hosts use stateless autoconfig for addresses.
Outgoing access list is V6-EGRESS-ACL-BI
Inbound common access list is not set, access list is V6-DENY-INGRESS-ACL
  Table Id is 0xe0800000
  Complete protocol adjacency: 0
  Complete glean adjacency: 0
  Incomplete protocol adjacency: 0
  Incomplete glean adjacency: 0

```

```
Dropped protocol request: 0
Dropped glean request: 0
```

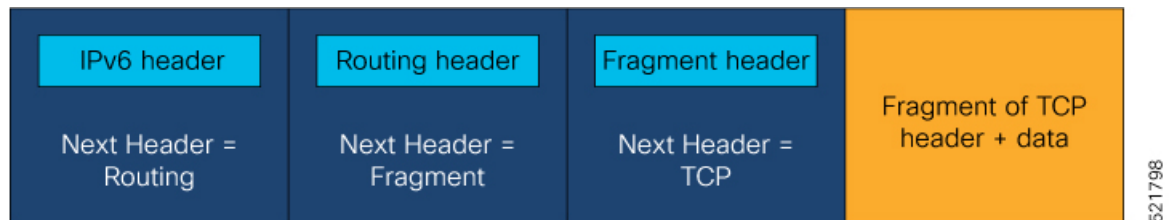
You have successfully configured ingress and egress IPv6 ACLs on a bundle interface.

Configuring IPv6 Extensions Headers in Ingress IPv6 Hybrid ACLs

Table 2: Feature History Table

Feature Name	Release Information	Description
IPv6 Extension Headers in Hybrid ACLs	Release 7.3.15	You can configure ingress IPv6 hybrid ACLs with compression level 2 to permit or deny packets on the basis of IPv6 extension headers set in them. IPv6 extension headers include routing headers, authentication headers, and destination option headers. These extension headers contain information that is used by network devices (routers and switches) to route or process an ingress IPv6 packet. The <code>ipv6 access-group</code> command is updated.

Figure 1: IPv6 Extension Header in an IPv6 Packet



As illustrated in the figure, an IPv6 packet may carry zero, one, or more extension headers, each identified by the Next Header field of the preceding header. To configure ingress Hybrid ACLs with ACEs that permit or deny packets on the basis of IPv6 extension headers, the IPv6 extension headers must be set immediately after the base IPv6 header as shown in the figure.

Therefore, you can filter ingress IPv6 packets through Hybrid ACLs with compression level 2 on the basis of IPv6 extension headers set in them.



Note You can filter ingress IPv6 packets having hop-by-hop extension headers through regular ACLs or through Hybrid ACLs with compression level 2.

Restrictions

- You cannot configure a protocol and an IPv6 extension header in the same ACE. However, you can create separate ACEs with protocols and extension header in a single ACL.
- You cannot filter egress IPv6 packets with Hybrid ACLs that have compression level 2 and have IPv6 extension headers set as an ACE.
- You cannot configure custom headers or mobility headers in an ACE filter ingress IPv6 packets through Hybrid ACLs with compression level 2.

Configuring Ingress IPv6 Hybrid ACLs Based on Extensions Headers

This following example shows you how to create an ingress IPv6 Hybrid ACL with compression level 2 based on IPv6 extensions headers:

```
/* Enter the global configuration mode and create an IPv6 Hybrid ACL with an ACE that has
routing extension header. */
Router# configure
Router(config)# ipv6 access-list ACL-EXT-HEADER
Router(config-ipv6-acl)# 10 deny ipv6 any any routing
Router(config-ipv6-acl)# 20 deny ospf any any
Router(config-ipv6-acl)# commit
Router(config-ipv6-acl)# exit

/* Enter the interface configuration mode and configure an ingress interface with the Hybrid
ACL twith compression level 2. */
Router(config)# interface hundredGigE 0/4/0/36
Router(config-if)# ipv6 access-group ACL-EXT-HEADER ingress compress level 2
Router(config-if)# commit
```

TCP Flags in ACLs

Table 3: Feature History Table

Feature Name	Release Information	Description
TCP Flags in Egress IPv6 ACLs	Release 7.3.15	<p>You can configure an egress IPv6 ACL that permits or denies packets on the basis of TCP flags set in them. Through this feature, you can restrict and manage traffic streams thereby enhancing network security.</p> <p>The following commands are updated:</p> <ul style="list-style-type: none"> • deny (IPv6) • permit (IPv6)

The transmission Control Protocol (TCP) is one of the most widely used protocol for data transmission in networks. The TCP header contains several one-bit boolean fields known as flags used to influence the flow of data across a TCP connection. TCP packets use TCP flags during a packet transfer to indicate connection state or provide additional information about the packet transfer. The various TCP flags are SYN, ACK, FIN, RST, URG, PSH, and EST. The following table provides information about these flags:

You can create ACEs that permit or deny packets based on TCP flags. By using ACLs:

- You can create ACEs that filter packets based on whether a packet has a TCP flag set or not.
- You can filter packets based on the presence or absence of any one TCP flag or combination of multiple TCP flags.

Therefore, ACLs based on TCP flags provide increased flexibility to filter packets and provides enhanced security. For example, ACLs can permit packets that have a SYN flag to ensure that the packets have a verified source.

TCP Flags

The following TCP flags can be present in a packet:

- SYN: Both the sender and receiver devices use the synchronisation (SYN) flag in only the first packet that is sent.
- ACK: The receiver devices use the acknowledgment (ACK) flag in the packet that is sent to acknowledge the successful receipt of a packet.
- FIN: The sender device uses the finished (FIN) flag in the last packet to indicate that there is no more data to be sent.
- RST: The receiver device uses the reset (RST) flag in the packet sent to the sender device when the receiver device receives a packet that is not expected.
- URG: The sender device uses the urgent (URG) flag in the packets to notify the receiver device to process the urgent packets before processing all other packets.
- PSH: The receiver device uses the push (PSH) flag that is similar to the URG flag and tells the receiver to process these packets as soon as they are received without waiting for any other packets to be received.
- EST: When a remote host receives TCP packets with a SYN flag set and if it does not support such a service, the remote host replies with an EST flag set in the packet. EST flag signifies both ACK and RST flags set in the packet.

Configuring ACLs Based on TCP Flags

You can use the **match-any** keyword in ACLs to permit or deny packets based on whether any of the configured TCP flags is set.

This following example shows you how to create an ACL to permit packets that have either the PSH or URG TCP flags set:

```
/* Enter the global configuration mode and create an ACL with name ACL-TCP.*/
Router# configure
Router(config)# ipv4 access-list ACL-TCP

/* Configure an ACL that permits packets with TCP flag that is either PSH or URG */
Router(config-ipv4-acl)# 10 permit tcp any any match-any + urg + psh
Router(config-ipv4-acl)# commit
```

This following example shows you how to create an ACL to permit packets that have either the SYN or ACK TCP flag set:

```
/* Enter the global configuration mode and create an ACL with name ACL-TCP. */
Router# configure
Router(config)# ipv4 access-list ACL-TCP
```

```
/* Configure an ACL that permits packets with both SYN and ACK TCP flags set. */
Router(config-ipv4-acl)# 10 permit tcp any any match-any + syn + ack
Router(config-ipv4-acl)# commit
```

This following example shows you how to create an ACL to permit packets that have the SYN flag set or the ACK flag not set:

```
/* Enter the global configuration mode and create an ACL with name ACL-TCP.*/
Router# configure
Router(config)# ipv4 access-list ACL-TCP

/* Configure an ACL that permits packets with the SYN flag set and the ACK flag not set.
*/
Router(config-ipv4-acl)# 10 permit tcp any any match-any + syn - ack
Router(config-ipv4-acl)# commit
```

You can use the **match-all** keyword in ACLs to permit or deny packets based on whether all the configured TCP flags are set.

This following example shows you how to create an ACL to permit packets that have both the URG flag and FIN flag set:

```
/* Enter the global configuration mode and create an ACL with name ACL-TCP.*/
Router# configure
Router(config)# ipv4 access-list ACL-TCP

/* Configure an ACL that permits packets with the SYN flag set and the ACK flag not set.
*/
Router(config-ipv4-acl)# 10 permit tcp any any match-all + urg + fin
Router(config-ipv4-acl)# commit
```

This following example shows you how to create an ACL to permit packets that have both the SYN flag set and the ACK flag not set:

```
/* Enter the global configuration mode and create an ACL with name ACL-TCP.*/
Router# configure
Router(config)# ipv4 access-list ACL-TCP

/* Configure an ACL that permits packets with the SYN flag set and the ACK flag not set.
*/
Router(config-ipv4-acl)# 10 permit tcp any any match-all + syn - ack
Router(config-ipv4-acl)# commit
```

Configuring Extended Access Lists

Use Extended Access Lists to verify more than just the source address of the packets. Attributes such as destination address, specific IP protocols, UDP or TCP port numbers, DSCP, and so on are validated. Traffic is controlled by a comparison of the attributes stated in the ACL with those in the incoming or outgoing packets.

Configuration Example

To configure Extended Access Lists, you must completed create an access list and specify the condition to allow or deny the network traffic.

```
/* Enter the global configuration mode and create the access list*/
Router# configure
Router(config)# ipv4 access-list acl_1
Router(config-ipv4-acl)# 10 remark Do not allow user1 to telnet out
```

```
/*Specify the condition to allow or deny the network traffic.*/
Router(config-ipv4-acl)# 10 permit 172.16.0.0 0.0.255.255
Router(config-ipv4-acl)# 20 deny 192.168.34.0 0.0.0.255
Router(config-ipv4-acl) commit
```

Running Configuration

```
Router#show running-config
Mon Jul 29 05:56:14.315 UTC
Building configuration...
!! IOS XR Configuration

!
ipv4 access-list acl_1
 10 permit ipv4 172.16.0.0 0.0.255.255 any
 20 deny ipv4 192.168.34.0 0.0.0.255 any
!
```

Verification

```
Router#show access-lists ipv4 acl_1 hardware ingress location 0/0/CPU0
Tue Jul 2 08:03:29.495 UTC
ipv4 access-list acl_1
66 deny igmp 30.0.20.0 0.0.0.255 30.0.10.0 0.0.0.255 v3-report (11604 matches)
67 deny igmp host 30.0.20.1 host 30.0.10.1 v2-report
```

IPv4 and IPv6 ACL in Class Map

Quality of Service (QoS) features are enhanced to support these:

- Support on L3 interface, sub-interface, bundle interface and bundle sub-interface
- Support for only ingress direction
- IPv6-supported match fields:
 - Destination Port
 - Fragment bit
 - ICMP type and code
 - IGMP type and code
 - IPv6 Destination Address
 - IPv6 Source Address
 - IPv6 Protocol
 - Precedence/DSCP
 - Source Port

Configuring IPv6 ACL QoS - An Example

This example shows how to configure IPv6 ACL QoS with IPv4 ACL and other fields :


```
ipv6 access-list aclv6
10 permit ipv6 1111:6666::2/64 1111:7777::2/64 authen
30 permit tcp host 1111:4444::2 eq 100 host 1111:5555::2
!

ipv4 access-list aclv4
10 permit ipv4 host 10.6.10.2 host 10.7.10.2
!

class-map match-any c.aclv6
match access-group ipv6 aclv6
match access-group ipv4 aclv4
match cos 1
end-class-map
!

policy-map p.aclv6
class c.aclv6
  set precedence 3
!
class class-default
!
end-policy-map
!
```

User-Defined TCAM Keys for IPv4 and IPv6

Access-lists use a TCAM (internal and external) to perform the lookup and action resolution on each packet. The TCAM is a valuable and constrained resource in hardware, which must be shared by multiple features. Therefore, the space (key width) available for these key definitions is also constrained. A key definition specifies which qualifier and action fields are available to the ACL feature when performing the lookup.

The key definitions are specific to a given ACL type, which can depend on the following attributes of the access-list:

- Direction of attachment only for ingress
- Protocol type (IPv4/IPv6)

Because the default key definitions are constrained (do not include all qualifier/action fields), User-Defined Key (UDK) definitions are supported for the following types:

- Traditional Ingress IPv4 ACL (uncompressed)
- Traditional Ingress IPv6 ACL (uncompressed)

The User-Defined TCAM Key (UDK) functionality provides the flexibility to define your own TCAM key for ingress, traditional, or IPv4 and IPv6 ACL only:

To include the well-known fields in the default TCAM key, see [IPv4 and IPv6 Key Formats, on page 27](#).

A User-Defined TCAM Key (UDK) can be defined globally or locally per line card. If both global and local UDK is available for a line card, then global UDK is ignored for the line card.

A UDK can be configured using the following command:

```
hw-module profile tcam format access-list [ipv4 | ipv6] field1 field2[location rack/slot/cpu0]
```

To define UDK globally, you can ignore the location option.

```
hw-module profile tcam format access-list [ipv4 | ipv6] field1field2
```



Note It is recommended to use global UDK for Cisco 8000 Series Routers Fixed platform.

User-Defined Fields

TCAM key consists of several qualifiers. Use the sets of qualifiers to filter packets for a given ACL. The User-Defined Field (UDF) allows you to define a custom qualifier by specifying the location and size of the field, using the following UDF command:

```
udf udf-name header [ inner | outer ] [13 | 14 ] offset byte-offset length no of bytes
```

You can add the UDF to a UDK as follows.

```
hw-module profile tcam format access-list [ipv4 | ipv6] qualifiers [udf1 udf-name udf2  
udf-name] [location rack/slot/cpu0]
```



Note You can define up to 8 UDFs systemwide. Currently, you can define UDFs globally.

Restrictions

- Cisco 8000 Series Routers support UDF only on L3 interfaces and not on L2 ports.
- Deep Packet Inspection is available only up to 128 Bytes inside the packet initiating from the L2 header.
- Cisco 8000 Series Routers support UDF only in ingress direction.
- Cisco 8000 Series Routers support only L3 and L4 base offsets for both inner and outer headers.
- Cisco 8000 Series Routers support only four bytes of match length.
- Modification for UDF configuration is allowed, but you must reload the related line card to be effective using the **reload location node Id** command.
- Modification for UDK is not supported using the **hw-module profile tcam format** command.
First, remove the existing UDK using the **no hw-module profile tcam format** command, then add a new UDK definition.
- If you configure UDK, you cannot use the default keys. But you can explicitly define the default fields in UDK.

Configuration Example

Steps

1. Create an UDF and define UDK.
2. Manually, reload the node on the line card.
3. Configure ACL using fields defined in UDK.

4. Attach ACL to an interface in ingress direction.

Configuration

The following example shows how to deny packets with the following condition:

- The packets with the source address as 192.0.2.0 and destination address as 203.0.113.0
- The packets with the payload pattern of 0x4567 at an offset of 48 bytes from the L3 header

```
/* Create an UDF and define global UDK and UDF*\
Router#configure terminal
Router(config)#udf udf_outer13_2b header outer 13 offset 48 length 2
Router(config)#hw-module profile tcam format access-list ipv4 ipv4-sip ipv4-dip udf1
udf_outer13_2b

In order to activate/deactivate this ipv4 profile, you must manually reload line cards
Router(config)#exit

/* Reload the node on the Cisco 8000 Series Routers line card*\
Router#reload location 0/8/CPU0
Wed Aug 21 14:12:40.123 UTC
Proceed with reload? [confirm]

Router#:Aug 21 14:12:44.120 UTC: fsdbagg[216]: %PKT_INFRA-FM-4-FAULT_MINOR : ALARM_MINOR
:FABRIC-PLANE-5 :DECLARE :: Fabric Plane-5 DOWN
Router:Aug 21 14:12:44.123 UTC: fsdbagg[216]: %PKT_INFRA-FM-4-FAULT_MINOR : ALARM_MINOR
:FABRIC-PLANE-6 :DECLARE :: Fabric Plane-6 DOWN

/*Configure ACL for User Defined Key*\
Router(config)#ipv4 access-list acl1
Router(config-ipv4-acl)10 deny ipv4 192.0.2.0 203.0.113.0 any udf udf_outer13_2b 0x4567
0xffff
Router(config-ipv4-acl)#exit
Router(config)#interface HundredGigE 0/0/0/24
Router (config-if)#ipv4 access-group acl1 ingress
```

The following example shows how to deny the IP in IP Tunneling packet with the following condition:

- Packet with the inner IP header having the DSCP as AF42.
- Packet with the payload pattern of 0x12345678 at offset of 20 bytes from outer layer 4 header.

```
Router#configure terminal
Router(config)#udf udf_inner13_1b header inner 13 offset 1 length 1
Router(config)#udf udf_outer14_4b header outer 14 offset 20 length 4
Router(config)#hw-module profile tcam format access-list ipv4 protocol udf1 udf_inner13_1b
udf udf_outer14_4b
```

IPv4 and IPv6 Key Formats

The following table shows the qualifier fields that are supported in the IPv4 and IPv6 key formats.



Note You cannot configure destination address and destination object group together for an ACL. Similarly, you cannot configure source address and source object group together for an ACL.

Table 4: Qualifier Fields Supported in IPv4 and IPv6 Key Formats

Parameter	Default TCAM Key	
	IPv4	IPv6
Destination Address	Supported	Supported
Destination Object Group	Supported	Supported
Destination Port	Supported	Supported
Fragment bit	Supported	Supported
Fragment offset	Supported	Not Supported
Fragment type	Supported	Not Supported
ICMP type and code	Supported	Supported
IGMP type and code	Supported	Supported
Packet Length	Supported	Supported
Protocol/Next Header	Supported	Supported
Precedence/DSCP	Supported	Supported
Source Object Group	Supported	Supported
Source Address	Supported	Supported
Source Port	Supported	Supported
TCP Flags	Supported	Supported
Time to live (TTL) Match	Supported	Not supported
UDF 1-8	Not supported	Not supported

The following table shows the action fields supported in the IPv4 and IPv6 key formats.

Table 5: Action Fields Supported in IPv4 and IPv6 Key Formats

Parameter	Default Action Field	
	IPv4	IPv6
Permit	Supported	Supported
Deny	Supported	Supported
Next Hop	Supported	Supported
Log	Supported for Ingress only	Supported for Ingress only

Parameter	Default Action Field	
	IPv4	IPv6
Capture	Supports only ingress Encapsulated Remote SPAN (ERSPAN)	Supports only ingress Encapsulated Remote Switch port Analyzer (ERSPAN)
Stats Counter	Deny stats is always enabled (permit stats is enabled by the hw-module profile stats acl-permit command)	Deny stats is always enabled (permit stats is enabled by the hw-module profile stats acl-permit command)

Modifying ACLs

This section describes a sample configuration for modification of ACLs.

```

*/ Create an Access List*/
Router(config)#ipv4 access-list acl_1

*/Add entries (ACEs) to the ACL*/
Router(config-ipv4-acl)#10 permit ip host 10.3.3.3 host 172.16.5.34
Router(config-ipv4-acl)#20 permit icmp any any
Router(config-ipv4-acl)#30 permit tcp any host 10.3.3.3
Router(config-ipv4-acl)#end

*/Verify the entries of the ACL*/:
Router#show access-lists ipv4 acl_1
ipv4 access-list acl_1
10 permit ip host 10.3.3.3 host 172.16.5.34
20 permit icmp any any
30 permit tcp any host 10.3.3.3

*/Add new entries, one with a sequence number "15" and another without a sequence
number to the ACL. Delete an entry with the sequence number "30":*/
Router(config)#ipv4 access-list acl_1
Router(config-ipv4-acl)# 15 permit 10.5.5.5 0.0.0.255
Router(config-ipv4-acl)# no 30
Router(config-ipv4-acl)# permit 10.4.4.4 0.0.0.255
Router(config-ipv4-acl)# commit

*/When an entry is added without a sequence number, it is automatically given a sequence
number
that puts it at the end of the access list. Because the default increment is 10, the entry
will have a sequence
number 10 higher than the last entry in the existing access list*/

*/Verify the entries of the ACL:*/
Router#show access-lists ipv4 acl_1
ipv4 access-list acl_1
 10 permit ipv4 host 10.3.3.3 host 172.16.5.34
15 permit 10.5.5.5 0.0.0.255---*/newly added ACE (with the sequence number)*/
20 permit icmp any any
30 permit ipv4 10.4.4.0 0.0.0.255 any ---*/newly added ACE (without the sequence number)*/

```

/The entry with the sequence number 30, that is, "30 permit tcp any host 10.3.3.3" is deleted from the ACL/

You have successfully modified ACLs in operation.

ACLs on Bridge Virtual Interfaces

Table 6: Feature History Table

Feature Name	Release Information	Description	
ACLs on BVI	Release 7.3.1	This feature allows traffic filtering by configuring ACLs on Bridge Virtual Interfaces (BVIs). A single configuration can be applied for multiple interfaces that are part of the BVI. You can therefore, filter traffic for a group of interfaces with a particular purpose.	

Bridge Virtual Interfaces (BVIs) provide a bridge between the routing and bridging domains on a router. You can configure BVI with an IP address that operates as a regular routed interface. You can configure an ACL on a BVI to filter the traffic for the network that uses the interface.

Increased TCAM Consumption with Configuring ACLs on BVIs

The consumption of TCAM resources is impacted in the following manner when ACLs are configured on BVIs.

- When an ACL is attached to a BVI interface, TCAM entries are programmed on all line cards regardless of physical interface membership. This process leads to greater consumption of TCAM resources even on line cards that do not have BVI member interfaces.
- For ingress and egress ACLs, the TCAM entries for the same ACL are shared across interfaces on the same NPU.

Restrictions for Configuring ACLs on BVIs

You must be aware of the following restrictions before proceeding to configure ACLs on BVIs.

- When you apply a security ACL on a BVI interface, TCAMs are programmed on all NPUs of all linecards. On a distributed system, ingress ACL stats on a BVI interface are seen on the line card where the L2 interface resides. Egress ACL stats are seen on the linecard of the L3 interface.

Configuration

The following section describes the procedure for configuring IPv4 ingress and egress ACLs on BVIs.

To configure IPv4 ingress and egress ACLs on a BVI, use the following procedure with a sample configuration.

1. Enter the Global Configuration mode, and configure an IPv4 ingress ACL.

```
Router(config)# ipv4 access-list v4-acl-ingress
Router(config-ipv4-acl)# 10 permit tcp any 10.1.1.0/24 dscp cs6
Router(config-ipv4-acl)# 20 deny udp any any eq ssh
Router(config-ipv4-acl)# 30 permit ipv4 any any
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# exit
```

2. Configure an IPv4 egress ACL.

```
Router(config)# ipv4 access-list v4-acl-egress
Router(config-ipv4-acl)# 10 deny ipv4 any any fragments log
Router(config-ipv4-acl)# 20 deny tcp any any ack
Router(config-ipv4-acl)# 30 permit ipv4 any any
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# exit
```

3. Configure the HundredGigE interface that you must to the BVI, and enable it for Layer 2 transport.

```
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# l2transport
Router(config-if-12)# commit
```

4. Attach the ingress and egress ACLs to the BVI.

```
Router(config)# interface BVI1
Router(config-if)# ipv4 access-group v4-acl-ingress ingress
Router(config-if)# ipv4 access-group v4-acl-egress egress
Router(config-if)# commit
Router(config-if)# exit
```

5. Configure the bridge domain with the HundredGigE interface and BVI.

```
Router(config)# l2vpn
Router(config-l2vpn)# bridge group BG1
Router(config-l2vpn-bg)# bridge-domain B1
Router(config-l2vpn-bg-bd)# interface HundredGigE 0/0/0/0
Router(config-l2vpn-bg-bd-ac)# routed interface BVI1
Router(config-l2vpn-bg-bd)# commit
Router(config-l2vpn-bg-bd)# exit
Router(config-l2vpn-bg)# exit
Router(config-l2vpn)# exit
```

6. Confirm that you have successfully committed the configuration.

```
Router(config)# show run
...
!
ipv4 access-list v4-acl-egress
 10 deny ipv4 any any fragments log
 20 deny tcp any any ack
 30 permit ipv4 any any
!
ipv4 access-list v4-acl-ingress
 10 permit tcp any 10.1.1.0/24 dscp cs6
 20 deny udp any any eq ssh
 30 permit ipv4 any any
```

```

!
interface HundredGigE 0/0/0/0
  l2transport
!
!
interface BVI1
  ipv4 address 209.165.200.224/27
  ipv4 access-group v4-acl-ingress ingress
  ipv4 access-group v4-acl-egress egress

!
!
l2vpn
  bridge group BG1
  bridge-domain B1
  interface HundredGigE 0/0/0/0
  !
  routed interface BVI1
  !
!
!
end

```

- Exit to the Executive Privileged mode and confirm that the ACLs are in operation.

```

Router# show access-lists interface bvi1
Tue May 9 10:01:25.732 EDT
Input ACL (common): HundredGigE 0/0/0/0 (interface): v4-acl-ingress
Output ACL: v4-acl-egress

Router# show access-lists summary
Tue May 9 10:02:01.167 EDT
ACL Summary:
Total ACLs configured: 2
Total ACEs configured: 6

Router# show access-lists ipv4 v4-acl-egress hardware egress location 0/0/CPU0
ipv4 access-list v4-acl-egress
10 deny ipv4 any any fragments log (15214 matches)
20 deny tcp any any ack (15214 matches)
30 permit ipv4 any any (15214 matches)

```

The output clearly shows the configured ACLs, the total number of ACEs (three per ACL), and also the ACE matches in hardware.

You have successfully configured and enabled IPv4 ingress and egress ACL on a BVI.

Configuring ACL-based Forwarding

Converged networks carry voice, video, and data. Users may need to route certain traffic through specific paths instead of using the paths computed by routing protocols. This is achieved by specifying the next-hop address in ACL configurations, so that the configured next-hop address from ACL is used for forwarding packet towards its destination instead of routing through packet-based destination address lookup. This feature of using next-hop in ACL configurations for forwarding is called ACL Based Forwarding (ABF).

ACL-based forwarding enables you to choose service from multiple providers for broadcast TV over IP, IP telephony, data, and so on, which provides a cafeteria-like access to the Internet. Service providers can divert user traffic to various content providers.

Restrictions

- Traffic outages can occur during transitions from an existing nexthop to another nexthop.
- IPv4 and IPv6 ABF nexthops routed over GRE interfaces are not supported.
- VRF-select (where only the VRF is configured for the nexthop) is not supported in ABF for IPv4 and IPv6 addresses.
- Logging of permit statistics for ABF is not supported.

Feature Highlights

- ABF supports nexthop modifications. You can modify a nexthop, remove a nexthop, or make changes between existing nexthops.



Note While defining an ACE rule, you must specify the VRF for all nexthops unless the nexthop is in the default VRF. This process ensures that the packets take the right path towards the nexthop.

- As ABF is ACL-based, packets that do not match an existing rule (ACE) in the ACL are subjected to the default ACL rule (drop all). If the ACL is being used for ABF-redirect only (not for security), then include an explicit ACE rule at the end of the ACL (lowest user priority) to match and "permit" all traffic. This ensures that all traffic that does not match an ABF rule is permitted and forwarded as normal.
- ABF is supported on permit rules only.
- ABF default route is not supported.
- Packets punted in the ingress direction from the NPU to the linecard CPU are not subjected to ABF treatment due to lack of ABF support in the slow path. These packets will be forwarded normally based on destination-address lookup by the software dataplane. Some examples of these types of packets are (but are not limited to) packets with IPv4 options, IPv6 extension headers, and packets destined for glean (unresolved/incomplete) adjacencies.

Configuration Example

To configure ACL-based forwarding for IPv4 packets, use the following steps:

1. Enter IPv4 access list configuration mode and configure an ACL.
2. Set the conditions for the ACL.
3. Configure nexthop addresses for ABF.

Configuration

To configure ACL-based forwarding for IPv4 packets, use the following configuration example:

```

/* Enter IPv4 access list configuration mode and configure an ACL: */
Router# configure
Router(config)# ipv4 access-list abf-acl

/* Set the conditions for the ACL and configure ABF: */
/* The next hop for this entry is specified. */
Router(config-ipv4-acl)# 10 permit ipv4 192.168.18.0 0.255.255.255 any nexthop1 ipv4 192.168.20.2
Router(config-ipv4-acl)# 15 permit ipv4 192.168.21.0 0.0.0.255 any
Router(config-ipv4-acl)# 20 permit ipv4 192.168.22.0 0.0.255.255 any nexthop1 ipv4 192.168.23.2
/* More than two nexthops */
Router(config-ipv4-acl)# 25 permit tcp any range 2000 3000 any range 4000 5000 nexthop1 ipv4 192.168.23.1 nexthop2 ipv4 192.168.24.1 nexthop3 ipv4 192.168.25.1

/* VRF support on ABF */
Router(config-ipv4-acl)# 30 permit tcp any eq www host 192.168.12.2 precedence immediate nexthop1 vrf vrf1_ipv4 ipv4 192.168.13.2 nexthop2 vrf vrf1_ipv4 ipv4 192.168.14.2

Router(config-ipv4-acl)# 35 permit ipv4 any any

Router(config-ipv4-acl)# commit

```

To configure ACL-based forwarding for IPv6 packets, use the following configuration example:

```

/* Enter IPv6 access list configuration mode and configure an ACL: */
Router# configure
Router(config)# ipv6 access-list abf-acl

/* Set the conditions for the ACL and configure ABF: */
/* The next hop for this entry is specified. */
Router(config-ipv6-acl)# 10 permit ipv6 2001:db8::/32 any nexthop1 ipv6 2001:db8::2

/* More than two nexthops */
Router(config-ipv6-acl)# 25 permit tcp any range 2000 3000 any range 4000 5000 nexthop1 ipv6 2001:db8::3 nexthop2 ipv6 2001:db8::4 nexthop3 ipv6 2001:db8::5

/* VRF support on ABF */
Router(config-ipv6-acl)# 30 permit tcp any eq www host 2001:db8::8 precedence immediate nexthop1 vrf vrf1_ipv6 ipv6 2001:db8::7 nexthop2 vrf vrf1_ipv6 ipv6 2001:db8::6

Router(config-ipv6-acl)# 35 permit ipv6 any any

Router(config-ipv6-acl)# commit

```

Running Configuration

```

Router# show access-lists ipv4
ipv4 access-list abf-acl
10 permit ipv4 192.168.18.0 0.255.255.255 any nexthop1 192.168.20.2
15 permit ipv4 192.168.21.0 0.0.0.255 any
20 permit ipv4 192.168.22.0 0.0.255.255 any nexthop1 192.168.23.2
25 permit tcp any range 2000 3000 any range 4000 5000 nexthop1 ipv4 192.168.23.1 nexthop2
ipv4 192.168.24.1 nexthop3 ipv4 192.168.25.1
30 permit tcp any eq www host 192.168.12.2 precedence immediate nexthop1 vrf vrf1_ipv4 ipv4
192.168.13.2 nexthop2 vrf vrf1_ipv4 ipv4 192.168.14.2
35 permit ipv4 any any
!
Router# show access-lists ipv6
ipv6 access-list abf-acl-ipv6
10 permit ipv6 2001:db8::/32 any nexthop1 ipv6 2001:db8::2
25 permit tcp any range 2000 3000 any range 4000 5000 nexthop1 ipv6 2001:db8::3 nexthop2
ipv6 2001:db8::4 nexthop3 ipv6 2001:db8::5
30 permit tcp any eq www host 2001:db8::8 precedence immediate nexthop1 vrf vrf1_ipv6 ipv6

```

```
2001:db8::7 nexthop2 vrf vrf1_ipv6 ipv6 2001:db8::6
35 permit ipv6 any any
```

Verification

Use the following command to verify the IP nexthop state in ABF to ensure that the expected nexthop is up:

```
Router# show access-lists ipv4 abf nexthops client pfilter_ea location 0/3/CPU0
Tue May 17 22:25:05.940 UTC
```

```
ACL name : abf-acl
ACE seq.      NH-1      NH-2      NH-3
-----
      20      Global 192.168.23.2      Not present      Not present
status                UP                Not present      Not present
exist                 No                Not present      Not present
pd ctx                Present          Not present      Not present
                    Track not present      Track not present      --
      25      Global 192.168.23.1      Global 192.168.24.1      Global 192.168.25.1
status                UP                UP                UP
exist                 Yes                Yes                Yes
pd ctx                Present          Present          Present
                    Track not present      Track not present      Track not present
```

Use the following command to verify if ABF is currently attached to any interfaces at any linecard:

```
Router# show access-lists usage pfilter location all
sh access-lists ipv4 abf nexthops client pfilter_ea loc 0/RP0/CPU0
Wed Jul 29 20:48:18.559 UTC
```

```
ACL name : abf-1
ACE seq.      NH-1      NH-2      NH-3
-----
      10      27.138.216.32      28.0.0.2      Not present
status                UP                UP                Not present
at status            Not Present      Not Present      Not present
exist                 No                Yes                Not present
vrf                  default          default          Not present
track                Not present      Not present      Not present
pd ctx                Present          Present          Not present
```

Associated Commands

- [ipv4 access-list](#)
- [ipv6 access-list](#)
- [show access-lists ipv4](#)
- [show access-lists ipv6](#)

Associated Topics

- [Configuring IPv4 ACLs](#)
- [Configuring IPv6 ACLs](#)

ABF over BVI

Table 7: Feature History Table

Feature Name	Release Information	Description
ACL-based Forwarding (ABF) on Bridge Virtual Interfaces (BVIs)	Release 7.5.3	This feature allows using Bridge Virtual Interfaces (BVIs) as next-hop in ABF. Here you can apply a single ABF configuration to multiple interfaces that are part of the BVI. You can therefore, divert user traffic for a group of interfaces with a particular purpose.

Starting with Cisco IOS XR Release 7.5.3, you can enable ACL-based Forwarding for IPv4 and IPv6 addresses on Bridge Virtual Interfaces (BVIs). With this, you could configure the ABF to forward incoming traffic on the BVIs matching the ACEs.

Restrictions

- ABF for IPv4 and IPv6 addresses over BVI is supported only in ingress direction.

Configuration

Perform the following steps to configure ABF over BVI.

1. Enter IPv4/IPv6 ACL configuration mode and configure an ACL.

```
/* IPV4 */
Router# configure
Router(config)# ipv4 access-list abf-acl1
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# exit

/* IPv6 */
Router# configure
Router(config)# ipv6 access-list abf-acl2
Router(config-ipv6-acl)# commit
Router(config-ipv6-acl)# exit
```

2. Create ACEs for the ACL and configure next hop addresses for ABF.

```
/* IPV4 */
Router# configure
Router(config)# ipv4 access-list abf-acl1
Router(config-ipv4-acl)# 10 permit ipv4 192.168.18.0 0.255.255.255 any nexthop1 ipv4
192.168.20.2
Router(config-ipv4-acl)# 15 permit ipv4 192.168.21.0 0.0.0.255 any
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# exit

/* IPv6 */
Router# configure
Router(config)# ipv6 access-list abf-acl2
Router(config-ipv6-acl)# 10 permit ipv6 2001:db8::/32 any nexthop1 ipv6 2001:db8::2
Router(config-ipv6-acl)# commit
Router(config-ipv6-acl)# exit
```

3. Configure the HundredGigE interface for the BVI, and enable it for Layer 2 transport.

```
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# l2transport
Router(config-if-l2)# commit
```

4. Enter the BVI mode and configure ABF.

```
Router(config)# int BVI 1
Router(config-if)# ipv4 access-group abf-acl1 ingress
Router(config-if)# ipv6 access-group abf-acl2 ingress
Router(config-if)# commit
```

5. Configure the bridge domain with the HundredGigE interface and BVI.

```
Router(config)# l2vpn
Router(config-l2vpn)# bridge group BG1
Router(config-l2vpn-bg)# bridge-domain B1
Router(config-l2vpn-bg-bd)# interface HundredGigE 0/0/0/0
Router(config-l2vpn-bg-bd-ac)# routed interface BVI1
Router(config-l2vpn-bg-bd)# commit
Router(config-l2vpn-bg-bd)# exit
Router(config-l2vpn-bg)# exit
Router(config-l2vpn)# exit
```

Running Configuration

```
Router(config)# show running configuration
!
ipv4 access-list abf-acl1
 10 permit ipv4 192.168.18.0 0.255.255.255 any nexthop1 ipv4 192.168.20.2
 15 permit ipv4 192.168.21.0 0.0.0.255 any
!
ipv6 access-list abf-acl2
 10 permit ipv6 2001:db8::/32 any nexthop1 ipv6 2001:db8::2
!
interface HundredGigE 0/0/0/0
 l2transport
!
!
interface BVI1
 ipv4 address 209.165.200.224/27
 ipv4 access-group abf-acl1 ingress
 ipv6 access-group abf-acl2 ingress
!
l2vpn
 bridge group BG1
  bridge-domain B1
   interface HundredGigE 0/0/0/0
    !
    routed interface BVI1
   !
  !
 !
end
```

Access Control List Counters

In Cisco IOS XR software, ACL counters are maintained both in hardware and software. Hardware counters are used for packet filtering applications. Software counters are used by all the applications mainly involving software packet processing.

Software counters are updated for the packets processed in software, for example, exception packets punted to the LC CPU for processing, or ACL used by routing protocols, and so on. The counters that are maintained are an aggregate of all the software applications using that ACL. To display software-only ACL counters, use the **show access-lists ipv4 access-list-name [sequence number]** command in EXEC mode.

ACL Statistics Counter

The ACL statistics counter feature allows you to track the count of packets that a router either permits or denies. A router permits or denies packets based on the ACL rules that you configure on a router interface. By default, the ACL statistics counter allows you to track only the the count of packets denied. By configuring the command, **hw-module profile stats acl-permit**, you can also track the count of packets that are permitted. Routers use this knowledge of the count of packets for ACL-based traffic mirroring. Support for ACL permit counters also allows you to track the ACE through which a router permits a packet.

Restrictions

- ACL-based forwarding (ABF) is not supported on a router after you configure the **hw-module profile stats acl-permit** command on that router.
- After you configure the **hw-module profile stats acl-permit** command on the router, based on the requirement, you must reload the router or the line cards. Configuring of the command followed by reloading the router or line cards enables the tracking of the permitted packet count on the router or line cards.

Configuration Example

To enable the tracking of the permitted packet count based on the ACL rules, use the following steps:

1. Enter global configuration mode and configure an ACL.
2. Enter interface configuration mode and attach the configured ACL on an interface.
3. Enable the tracking of the permitted packet count.
4. Based on the requirement, reload the router or line cards.

Configuration

```
/* Enter global configuration mode and configure an ACL. */
Router# configure
Router(config)# ipv4 access-list TEST
Router(config-ipv4-acl)# 10 permit ipv4 any any
Router(config-ipv4-acl)# 20 deny udp any any
Router(config-ipv4-acl)# commit
Router(config)# exit
```

```

/* Enter the interface configuration mode and attach the configured ACL on an interface.
*/
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# ipv4 access-group TEST ingress
Router(config-if)# commit
Router(config-if)# exit

/* Enable the tracking of the permitted packet count. */
Router(config)# hw-module profile stats acl-permit
Router(config)# commit

/* Reload the line cards. */
Router# reload location 0/0/CPU0

```

Running Configuration

```

Router# show running-configuration
ipv4 access-list TEST
10 permit ipv4 any any
20 deny udp any any
!
interface HundredGigE 0/0/0/0
ipv4 access-group TEST ingress
!
hw-module profile stats acl-permit
end

```

Verification

Use the **show access-lists ipv4 *acl-name* hardware ingress location *node-id*** command to verify that you have enabled or disabled the tracking of the permitted packet count.

```

Router# show access-lists ipv4 TEST hardware ingress location 0/0/CPU0
ipv4 access-list test-v4-ttl
 10 permit ipv4 any any ttl eq 100
 20 deny ipv4 any any ttl eq 200 (44043 matches)
Router# configure
Router(config)# hw-module profile stats acl-permit
In order to activate/deactivate this stats profile, you must manually reload the chassis/all
line cards
Router(config)# commit
Router# reload location 0/0/CPU0

```

```

Router# show access-lists ipv4 TEST hardware ingress location 0/0/CPU0
ipv4 access-list test-v4-ttl
 10 permit ipv4 any any ttl eq 100 (346318 matches)
 20 deny ipv4 any any ttl eq 200 (44043 matches)

```

Configuring ACLs with Fragment Control

The non-fragmented packets and the initial fragments of a packet were processed by IP extended access lists (if you apply this access list), but non-initial fragments were permitted, by default. However, now, the IP Extended Access Lists with Fragment Control feature allows more granularity of control over non-initial fragments of a packet. Using this feature, you can specify whether the system examines non-initial IP fragments of packets when applying an IP extended access list.

As non-initial fragments contain only Layer 3 information, these access-list entries containing only Layer 3 information, can now be applied to non-initial fragments also. The fragment has all the information the system requires to filter, so the access-list entry is applied to the fragments of a packet.

This feature adds the optional **fragments** keyword to the following IP access list commands: **deny** and **permit**. By specifying the **fragments** keyword in an access-list entry, that particular access-list entry applies only to non-initial fragments of packets; the fragment is either permitted or denied accordingly.

The behavior of access-list entries regarding the presence or absence of the **fragments** keyword can be summarized as follows:

If the Access-List Entry has...	Then...
...no fragments keyword and all of the access-list entry information matches	For an access-list entry containing only Layer 3 information: <ul style="list-style-type: none"> • The entry is applied to non-fragmented packets, initial fragments, and non-initial fragments. For an access-list entry containing Layer 3 and Layer 4 information: <ul style="list-style-type: none"> • The entry is applied to non-fragmented packets and initial fragments. <ul style="list-style-type: none"> • If the entry matches and is a permit statement, the packet or fragment is permitted. • If the entry matches and is a deny statement, the packet or fragment is denied. • The entry is also applied to non-initial fragments in the following manner. Because non-initial fragments contain only Layer 3 information, only the Layer 3 portion of an access-list entry can be applied. If the Layer 3 portion of the access-list entry matches, and <ul style="list-style-type: none"> • If the entry is a permit statement, the non-initial fragment is permitted. • If the entry is a deny statement, the next access-list entry is processed. <p>Note The deny statements are handled differently for non-initial fragments versus non-fragmented or initial fragments.</p>
...the fragments keyword and all of the access-list entry information matches	The access-list entry is applied only to non-initial fragments. <p>Note If the fragments keyword is configured for an access-list entry, the Layer 4 information will be ignored for the non-initial fragments.</p>

You should not add the **fragments** keyword to every access-list entry, because the first fragment of the IP packet is considered a non-fragment and is treated independently of the subsequent fragments. Because an initial fragment will not match an access list permit or deny entry that contains the **fragments** keyword, the packet is compared to the next access list entry until it is either permitted or denied by an access list entry that does not contain the **fragments** keyword. Therefore, you may need two access list entries for every deny entry. The first deny entry of the pair will not include the **fragments** keyword, and applies to the initial

fragment. The second deny entry of the pair will include the **fragments** keyword and applies to the subsequent fragments. In the cases where there are multiple **deny** access list entries for the same host but with different Layer 4 ports, a single deny access-list entry with the **fragments** keyword for that host is all that has to be added. Thus all the fragments of a packet are handled in the same manner by the access list.

Packet fragments of IP datagrams are considered individual packets and each fragment counts individually as a packet in access-list accounting and access-list violation counts.



Note The **fragments** cannot be configured for an access-list entry that contains any Layer 4 information.



Note Within the scope of ACL processing, Layer 3 information refers to fields located within the IPv4 header; for example, source, destination, protocol. Layer 4 information refers to other data contained beyond the IPv4 header; for example, source and destination ports for TCP or UDP, flags for TCP, type and code for ICMP.

Configuring an IPv4 ACL to Match on Fragment Type

Most DoS (Denial of Service) attacks work by flooding the network with fragmented packets. By filtering the incoming fragments of the packet in a network, an extra layer of protection can be added against such attacks.



Note IPv6 Extended Access Lists do not support the configuration of fragment types.

You can configure an IPv4 ACL to match on the fragment type, and perform an appropriate action. You can use the following sample configuration with the different fragment options:

```
/* Enter the global configuraton mode and configure an IPv4 access list */
Router# config
Router(config)# ipv4 access-list TEST
Router(config-ipv4-acl)# 10 permit tcp any any

/* Configure an ACE to match on the dont-fragment flag (indicates a non-fragmented packet)
and forward the packet to the default (pre-configured) next hop */
Router(config-ipv4-acl)# 20 permit tcp any any fragment-type dont-fragment default

/* Configure an ACE to match on the is-fragment flag (indicates a fragmented packet)
and forward the packet to a next hop of 10.10.10.1 */
Router(config-ipv4-acl)# 30 permit udp any any fragment-type is-fragment nexthop1 ipv4
10.10.10.1

/* Configure an ACE to match on the first-fragment flag (indicates the first fragment of a
fragmented packet)
and forward the packet to a next hop of 20.20.20.1 */
Router(config-ipv4-acl)# 40 permit ospf any any fragment-type first-fragment nexthop1 ipv4
20.20.20.1

/* Configure an ACE to match on the last-fragment flag (indicates the last fragment of a
fragmented packet)
and forward the packet to a next hop of 30.30.30.1 */
```

```
Router(config-ipv4-acl)# 50 permit icmp any any fragment-type last-fragment nexthop1 ipv4
30.30.30.1
Router(config-ipv4-acl)# commit
```

Use Case: Configuring an IPv4 ACL to Match on the First Fragment and Last Fragment

This section describes an use case, where you configure an ACL to forward a fragment if it is the first fragment of the packet and discard a fragment if it is the last fragment of the packet.

In this configuration, the ACL checks the fragment offset value ('0' for the first fragment). If the fragment is the first fragment of the packet, the packet is forwarded. If the fragment is the last fragment of the packet, it is dropped at the interface.

```
/* Enter the global configuraton mode and configure an IPv4 access list */
Router# config
Thu Jan 11 11:56:27.221 IST
Router(config)# ipv4 access-list ACLFIRSTFRAG

/* Configure an ACE to match on the first fragment.
If the fragment offset value equals 0, the fragment is forwarded to the 192.168.1.2 next
hop */
Router(config-ipv4-acl)# 10 permit tcp any any fragment-type first-fragment nexthop1 ipv4
192.168.1.2

/* Configure an ACE to match on the last fragment, and drop the fragment at the interface.
*/
Router(config-ipv4-acl)# 20 deny tcp any any fragment-type last-fragment
Router(config-ipv4-acl)# commit
Thu Jan 11 12:01:33.297 IST

/* Validate the configuration */
Router(config-ipv4-acl)# do show access-lists
Thu Jan 11 12:05:23.646 IST
ipv4 access-list ACLFIRSTFRAG
 10 permit tcp any any fragment-type first-fragment nexthop1 ipv4 192.168.1.20
 20 deny tcp any any fragment-type last-fragment
```

You have successfully configured an IPv4 ACL to match on the fragment type.

Matching by Fragment Offset in ACLs

You can configure an access control list (ACL) rule to filter packets by the fragment-offset value. Depending on whether a packet matches the criteria in a permit or deny statement, the packet is either processed or dropped respectively at the interface. Fragment-offset filtering is supported only on ingress direction with compression mode of an ACL.



Note

- Fragment offset filtering is supported for IPv4 packets in the default TCAM key for NC57-18DD-SE line cards, in traditional ACL mode, and not in compressed ACL mode.
- IPv6 Extended Access Lists do not support the configuration of ACL matching by fragment-offset values.

Configuring ACL Matching by Fragment Offset

To configure fragment-offset match in ACL, use the **fragment-offset** option in **permit** or **deny** command in IPv4 or IPv6 access-list configuration mode.



Note For fragment-offset filtering, you must attach the particular ACL to an interface with compression level 2. Else, the configuration is rejected.

Configuration

This example shows how to specify an ACL rule based on the fragment-offset per IPv4 header. Here, the packet is permitted only if the fragment-offset in the IPv4 header of the packet is within the range of 300-400. The value *300-400* is based on the 8-byte unit, which is same as fragment-offset of *2400-3200* bytes.

Running Configuration

```
ipv4 access-list fragment-offset-acl
 10 permit ipv4 any any fragment-offset range 300 400
!
```

Verify Fragment-offset Match in ACL

```
Router# show access-lists ipv4 fragment-offset-acl usage pfilter loc 0/4/CPU0
```

```
Wed Apr 12 19:49:54.457 UTC
Interface : Bundle-Ether70
  Input ACL : Common-ACL : N/A  ACL : fragment-offset-acl  (comp-lvl 3)
  Output ACL : N/A
```

```
Router# show access-lists ipv4 fragment-offset-acl hardware ing int Bundle-Ether70 loc
0/4/CPU0
```

```
Wed Apr 12 19:51:07.837 UTC
ipv4 access-list fragment-offset-acl
 10 permit ipv4 any any fragment-offset range 300 400
```

Configuring ACL Filtering by IP Packet Length

You can configure an access control list to filter packets by the packet length at an ingress interface. Depending on whether a packet matches the packet-length condition in a permit or deny statement, the packet is either processed or dropped respectively at the interface.

To configure packet length filtering in ACL, use the **packet-length** option in **permit** or **deny** command in IPv4 or IPv6 access-list configuration mode.

Restrictions

Packet length filtering feature in ACL is subjected to these restrictions:

- Packet length filtering is supported only on ingress direction, for both traditional (non-compression) and hybrid (compression) ACLs.
- IPv6 packet length filtering is supported only for hybrid ACLs; not for traditional ACLs.
- Only quantized (value divisible by 16) packet length filtering is supported for traditional ACLs on IPv4.
- For ACLs with compression level 2, by using the `hw-module profile tcam format` command, you can add packet length as a User Defined Field (UDF) along with **src-object-group** and **dst-object-group** as other UDFs.
- For ACLs with compression level 0, by defining packet length along with `src-object-group` and `dst-object-group` as a UDF, you can add packet length as a User Defined Field (UDF) along with **src-addr** and **dst-addr** as other UDFs. For more information, see [User-Defined Fields, on page 26](#).

Configuring Scaled IPv4 ACLs to Filter by Packet Length

To configure a scaled ACL to filter by packet length in IPv4 networks, use the following steps.

1. Enable packet length filtering in the global configuration mode by using the `hw-module` command.

```
Router# config
Router(/config)# hw-module profile tcam format access-list ipv4 src-object-group
dst-object-group dst-port proto packet-length frag-bit port-range
```

2. Enter the global configuration mode and create an object group for configuring a scaled ACL.

```
Router(config)# object-group network ipv4 netobject1
Router(config-object-group-ipv4)# 50.0.0.0/24
Router(config-object-group-ipv4)# commit
```

3. From the global configuration mode, configure an IPv4 access list to filter packets by the packet length value.

In this particular example, we configure a statement to process only those packets that match the specified packet length condition. All other packets are dropped when this ACL is applied to an ingress interface.

```
Router# configure
Router(config)# ipv4 access-list scaled_acl1
Router(config-ipv4-acl)# 10 permit ipv4 net-group netobject1 any packet-length eq 1000
```

4. Commit the ACL and exit the IPv4 ACL configuration mode.

```
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# end
```

5. Apply the ACL to the required HundredGigE interface.

```
Router(config)# interface HundredGigE 0/5/0/3
Router(config-if)# ipv4 access-group scaled_acl1 ingress compress level 2
```

6. Commit the configuration and exit the interface configuration mode.

```
Router(config-if)# commit
Router(config-if)# end
```

7. Verify your configuration.

```
Router# show access-lists scaled_acl1
ipv4 access-list scaled_acl1
10 permit ipv4 net-group netobject1 any packet-length eq 1000
```

8. Verify the ACL matches in hardware.

```
Router# show access-lists scaled_acl1 hardware ingress location 0/5/CPU0
ipv4 access-list scaled_acl1
10 permit ipv4 net-group netobject1 any packet-length eq 1000 (1500 hw matches)
```

You have successfully configured a scaled IPv4 ACL to filter by packet length.

Configuring Scaled IPv6 ACLs to Filter by Packet Length

To configure a scaled ACL to filter by packet length in IPv6 networks, use the following steps.

1. Enable packet length filtering in the global configuration mode by using the `hw-module` command.

```
Router# config
Router(/config)# hw-module profile tcam format access-list ipv4 src-object-group
dst-object-group dst-port proto packet-length frag-bit port-range
```

2. Enter the global configuration mode and create an object group for configuring a scaled ACL.

```
Router(config)# object-group network ipv6 netobject2
Router(config-object-group-ipv6)# 2001::0/128
Router(config-object-group-ipv6)# commit
```

3. From the global configuration mode, configure a scaled IPv6 access list to filter packets by the packet length value.

In this particular example, we configure a statement to process only those packets that match the specified packet length condition. All other packets are dropped when this ACL is applied to an ingress interface.

```
Router(config)# ipv6 access-list scaled_acl2
Router(config-ipv6-acl)# 10 permit ipv6 net-group netobject2 any packet-length eq 1000
Router(config-ipv6-acl)# commit
```

4. Commit the ACL and exit the IPv6 ACL configuration mode.

```
Router(config-ipv6-acl)# commit
Router(config-ipv6-acl)# end
```

5. Apply the ACL to the required HundredGigE interface.

```
Router# config
Router(config)# interface HundredGigE 0/5/0/3
Router(config-if)# ipv6 access-group scaled_acl2 ingress compress level 3
```

6. Commit the configuration and exit the interface configuration mode.

```
Router(config-if)# commit
Router(config-if)# end
```

7. Verify your configuration.

```
Router# show access-lists ipv6 scaled_acl2
ipv6 access-list scaled_acl2
10 permit ipv6 net-group netobject2 any packet-length eq 1000
```

8. Verify the ACL matches in hardware.

```
Router# show access-lists ipv6 scaled_acl2 hardware ingress location 0/5/CPU0
ipv6 access-list scaled_acl2
10 permit ipv6 net-group netobject2 any packet-length eq 1000 (2000 hw matches)
```

You have successfully configured a scaled IPv6 ACL to filter by packet length.

Configuring TTL Matching

You can configure ACLs to match on the TTL value specified in the IPv4 header. You can specify the TTL match condition to be based on a single value, or multiple values.

TTL matching is supported for both ingress and egress ACLs.

Configuration

Use the following steps to configure TTL matching.

```
/* Configure an IPv4 ACL with the TTL parameters */
Router(config)# ipv4 access-list acl-v4
Router(config-ipv4-acl)# 10 deny tcp any any ttl eq 100
Router(config-ipv4-acl)# 20 permit tcp any any ttl range 1 50
Router(config-ipv4-acl)# 30 permit tcp any any ttl neq 100
Router(config-ipv4-acl)# commit
Thu Nov  2 12:22:58.948 IST

/* Attach the IPv4 ACL to the HundredGigE interface */
Router(config)# interface HundredGigE 0/0/0/0
Router(config-if)# ipv4 address 15.1.1.1 255.255.255.0
Router(config-if)# ipv4 access-group acl-v4 ingress
Router(config-if)# commit
```

Running Configuration

Validate your configuration by using the **show run** command.

```
Router(config)# show run
Thu Nov  2 14:01:53.376 IST
Building configuration...
!! IOS XR Configuration 0.0.0
!! Last configuration change at Thu Nov  2 12:22:59 2017 by annseque
!
ipv4 access-list acl-v4
 10 deny tcp any any ttl eq 100
 20 permit tcp any any ttl range 1 50
 30 permit tcp any any ttl neq 100
!
interface HundredGigE 0/0/0/0
  ipv4 address 15.1.1.1 255.255.255.0
```

```
ipv4 access-group acl-v4 ingress
!
```

You have successfully configured TTL matching for IPv4 ACLs.

Understanding IP Access List Logging Messages

Cisco IOS XR software can provide logging messages about packets permitted or denied by a standard IP access list. That is, any packet that matches the access list causes an informational logging message about the packet to be sent to the console. The level of messages logged to the console is controlled by the **logging console** command in global configuration mode.

The first packet that triggers the access list causes an immediate logging message, and subsequent packets are collected over 5-minute intervals before they are displayed or logged. The logging message includes the access list number, whether the packet was permitted or denied, the source IP address of the packet, and the number of packets from that source permitted or denied in the prior 5-minute interval.

However, you can use the { **ipv4 | ipv6** } **access-list log-update threshold** command to set the number of packets that, when they match an access list (and are permitted or denied), cause the system to generate a log message. You might do this to receive log messages more frequently than at 5-minute intervals.



Caution If you set the *update-number* argument to 1, a log message is sent right away, rather than caching it; every packet that matches an access list causes a log message. A setting of 1 is not recommended because the volume of log messages could overwhelm the system.

Even if you use the { **ipv4 | ipv6** } **access-list log-update threshold** command, the 5-minute timer remains in effect, so each cache is emptied at the end of 5 minutes, regardless of the number of messages in each cache. Regardless of when the log message is sent, the cache is flushed and the count reset to 0 for that message the same way it is when a threshold is not specified.



Note The logging facility might drop some logging message packets if there are too many to be handled or if more than one logging message is handled in 1 second. This behavior prevents the router from using excessive CPU cycles because of too many logging packets. Therefore, the logging facility should not be used as a billing tool or as an accurate source of the number of matches to an access list.

Per Interface Statistics

When binding ACL to interfaces, you can configure ACE drop counts by using the **interface-statistics** keyword in the per-interface mode.

In Cisco 8000 Series Routers, you can allocate up to 8 stats counters per NPU.

This also limits the number of the interface that can support the same ACL in per-interface-stats mode to 8. Additional binding of the same ACL in per-interface-stats mode will be rejected.



Note For a specific direction, all interfaces on a line card using the same ACL must be configured in the same stats mode, if not the subsequent binding will be rejected.

This is true for the bundle interface too. If you add any member to the existing bundle interface with a different stats mode, the binding will be rejected.

Configuration Example

```
Router# interface HundredGigE 0/0/0/0
Router(config-if)#ipv4 address 1.1.1.1 255.255.0.0
Router(config-if)#ipv6 address 2001::1/64
Router(config-if)#ipv4 access-group test ingress interface-statistics
Router(config-if)#commit
```

Verification

To display deny stats on the interface:

```
Router#show access-lists ipv4 test hardware ingress interface
FourHundredGigE 0/1/0/0 location 0/1/CPU0
```

```
ipv6 access-list test
10 permit ipv6 any 200:23::/64
20 deny udp any any (2356 matches)
```