

Quality of Service Configuration Guide

First Published: 2025-09-15

Last Modified: 2026-07-17

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: https://www.cisco.com/c/en/us/about/legal/trademarks.html. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2025 Cisco Systems, Inc. All rights reserved.



CONTENTS

CHAPTER 1

Quality of service 1

Feature history for Quality of Service 1

Introduction to QoS 2

Modular QoS CLI 2

Benefits of using QoS 3

QoS mechanisms 4

Traffic conditioning 4

Classification 4

Metering 14

Marking 14

Policing 16

Shaping 17

Queuing and scheduling 18

Bandwidth 18

Priority queues 18

Queue buffers 20

Weighted random early detection 20

Trust behavior 21

Why is trust behavior important? 21

Where is trust behavior configured? 21

Common trust options 21

How QoS works 22

Differentiated Services architecture 22

Layer 3 packet prioritization bits 23

How QoS uses classification 23

Packet classification 24

QoS prerequisites 26
QoS limitations 27
Limitations on wired targets 27
Limitations on EtherChannel and channel member interfaces 29
Limitations on egress queuing policy 30
Supported actions in queuing policy 30
Configure Class, Policy, and Maps 31
Create a traffic class 31
Create a traffic policy 33
Create a traffic queueing policy 35
Configure class-based packet marking for ingress policy-map 37
Attach a traffic policy to an interface 44
Classify, police, and mark traffic on physical ports by using policy maps 45
Classify and mark traffic by using policy maps 49
Configure table maps 51
Configure QoS functions 54
Configure bandwidth 54
Configure police 55
Configure priority 57
Configure queues and shaping 59
Configure egress queue characteristics 59
Configure queue limits 59
Configure shaping 61
Configure shaper profile queuing 62
Monitor QoS 63
Examples: TCP protocol classification 64
Examples: UDP protocol classification 65
Examples: RTP protocol classification 66
Examples: Classification by ACLs 66
Examples: Class of service Layer 2 classification 67
Examples: Class of service DSCP classification 67
Examples: Classification by DSCP or precedence values 67
Examples: Classification for voice and video 68
Examples: Queue-limit configuration 69

Examples: Policing action configuration 70
Examples: Policing units 71
Examples: Single-rate two-color policing configuration 71
Examples: Dual-rate three-color policing configuration 71
Examples: Table map marking configuration 71
Display QoS configuration 72

CHAPTER 2 Auto-QoS 81
Feature history for Auto-QoS 81
Auto-QoS overview 82
Auto-QoS compact overview 82
Limitations for Auto-QoS 83
How Auto-QoS generates QoS templates 84
Auto-QoS policy and class maps 84
Behavior when Auto-QoS is enabled 85
Behavior when Auto-QoS compact is enabled 85
How to configure Auto-QoS 87
Upgrade Auto-QoS 88
Enable Auto-QoS compact 90
Monitor Auto-QoS 92
Troubleshoot Auto-QoS 92
Configuration examples 93
Example: auto qos trust cos 93
Example: auto qos trust dscp 96
Example: auto qos video cts 102
Example: auto-qos video ip-camera 105
Example: auto qos video media-player 110
Example: auto qos voip trust 115
Example: auto qos voip cisco-phone 119
Example: auto qos voip cisco-softphone 123
Example: auto qos classify police 129
Example: auto qos global compact 134

CHAPTER 3 HQoS 139

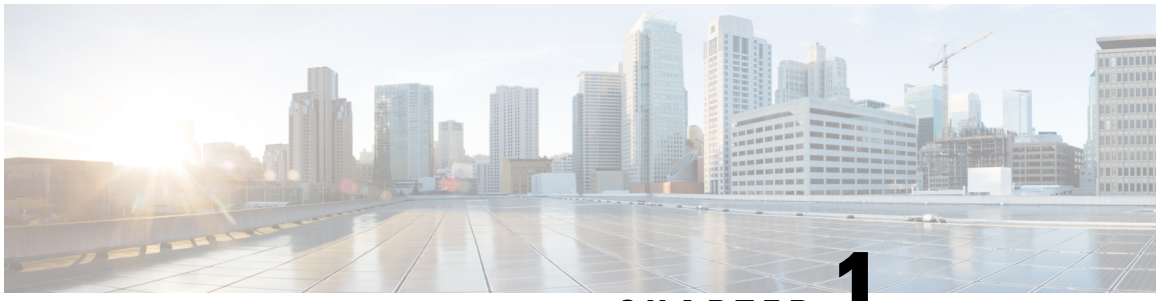
Feature History for HQoS 139
Information About HQoS 139
Information About HQoS 139
Prerequisites for HQoS 140
Restrictions for HQoS 141
Limitation for HQoS 141
How to Configure HQoS 141
Configuration Samples for HQoS 141

CHAPTER 4 **Layer 3 subinterface queuing 147**
Feature history for Layer 3 Subinterface Queuing 147
Default queuing mode 147
Subinterface propagation queuing mode 148
Hierarchical QoS 149
HQoS configuration options 150
Limitations for Layer 3 subinterface queuing 150
Enabling subinterface queuing policy 151
Enabling subinterface priority propagation mode 154
Configuring hierarchical QoS policy on a subinterface 155
Configuration examples 158
Monitor Layer 3 subinterface queuing 160

CHAPTER 5 **MPLS QoS 161**
Feature history for MPLS QoS 161
MPLS QoS 161
MPLS experimental field 162
Benefits of MPLS EXP classification and marking 163
Prerequisites for MPLS QoS 163
Restrictions for MPLS QoS 163
Configure MPLS QoS 164
Classify MPLS encapsulated packets 164
Mark MPLS EXP on the outermost label 165
Mark MPLS EXP on label switched packets 166
Configure conditional marking 167

Configuration examples 168
Example: Classify MPLS encapsulated packets 169
Example: Mark MPLS EXP on outermost label 169
Example: Mark MPLS EXP on label-switched packets 170
Example: Configure conditional marking 170

CHAPTER 6 **Application Visibility and Control QoS 173**
Feature History for Application Visibility and Control 173
Application Visibility and Control 174
Restrictions for Application Visibility and Control 175
Supported AVC class map and policy map formats 176
Match and collect fields in a wired AVC monitor 178
NBAR2 custom applications 179
Customize local applications using NBAR2 180
NBAR2 dynamic hitless protocol pack upgrade 182
Load the NBAR2 protocol pack 182
Configure Application Visibility and Control 183
Enable application recognition on an interface 183
Configure Application Visibility and Control QoS 184
Configure wired AVC flexible Netflow 186
Create a flow record 186
Create a flow exporter 191
Create a flow monitor 192
Associate flow monitor to an interface 194
Monitor Application Visibility and Control 194
Verify Application Visibility and Control 195
Troubleshooting guidelines for Application Visibility and Control 205



CHAPTER 1

Quality of service

This document provides detailed understanding about QoS, various components and their configuration.

- Feature history for Quality of Service, on page 1
- Introduction to QoS, on page 2
- Benefits of using QoS, on page 3
- QoS mechanisms, on page 4
- How QoS works, on page 22
- QoS prerequisites, on page 26
- QoS limitations, on page 27
- Configure Class, Policy, and Maps, on page 31
- Configure QoS functions, on page 54
- Configure queues and shaping, on page 59
- Monitor QoS, on page 63

Feature history for Quality of Service

This table provides release and platform support information for the features explained in this module.

These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Release	Feature name and description	Supported platform
Cisco IOS XE 26.2.1ea	Quality of Service: QoS feature support has been introduced.	Cisco C9550 Series Smart Switches
Cisco IOS XE 17.18.1	Quality of Service: QoS provides preferential treatment to specific types of traffic at the expense of other traffic types. Without QoS, the device offers best-effort service for each packet, regardless of the packet contents or size.	Cisco C9350 Series Smart Switches Cisco C9610 Series Smart Switches

Introduction to QoS

Quality of Service (QoS) refers to enhancing network service for selected traffic across technologies like Ethernet, Frame Relay, and IP-routed networks. You can use QoS to ensure that your applications receive predictable service levels, including data throughput capacity (bandwidth), latency variations (jitter), and delay.

QoS manages network traffic to ensure the performance of critical applications, especially when network capacity is limited. Essentially, QoS lets network administrators differentiate traffic and apply behaviors to ensure optimal performance for essential applications.

A default policy is always present for the device. This policy sets the traffic class and discard class. Without QoS, the device offers a best effort service for each packet, irrespective of its contents or size.

Modular QoS CLI

Modular QoS CLI (MQC) is a structured and flexible framework used on Cisco IOS and other network operating systems to configure QoS policies. MQC provides a consistent and modular approach to define, apply, and manage QoS features on network devices like routers and switches.

The MQC framework is built around three main components:

- Class map (class-map):
  - Purpose: This is where traffic classification happens. A class map defines the criteria used to identify and group specific types of network traffic.
  - Mechanism: You define match criteria within a class map. Common match criteria include:
    - Access Control Lists (ACLs) - example, match access-group 101
    - Differentiated Services Codepoint (DSCP) values - example, match dscp ef
    - IP Precedence values - example, match ip precedence critical
    - Protocol types - example, match cos (for Layer 2 packets)
  - Output: A class map essentially creates a "class" or "category" for traffic that matches its defined criteria.
- Policy map (policy-map):
  - Purpose: This is where traffic conditioning actions are defined for each class of traffic. A policy map links the classified traffic (from the class maps) to specific QoS actions.
  - Mechanism: Inside a policy map, you reference one or more class maps using the class command. For each class, you specify the QoS actions to be applied to the traffic belonging to that class. These actions can include:
    - Bandwidth allocation: bandwidth (guaranteed), priority (strict priority queue)
    - Congestion management: queue-limit
    - Congestion avoidance: random-detect (WRED)
    - Traffic shaping: shape average

- Traffic policing: police
- Marking: set dscp, set ip precedence, set cos (to re-mark traffic), set traffic-class <0-7>
- Dropping: drop (implicitly or explicitly)

- Output: A policy map defines a set of rules and actions that describe how different types of traffic should be treated.

- Service policy (service-policy):
  - Purpose: This is where the defined QoS policy is applied to a specific interface or sub-interface (or support on port-channel).
  - Mechanism: You use the service-policy command to attach a policy map to an interface. You specify whether the policy should be applied to inbound (input) or outbound (output) traffic.
  - Output: Once applied, the network device starts classifying traffic according to the class maps within the referenced policy map and applies the corresponding QoS actions defined in that policy map to the traffic passing through that interface.

How MQC links to traffic conditioning and classification

- Classification: The class-map component directly implements the classification function of traffic conditioning by defining the criteria to identify different traffic flows.
- Conditioning actions: The policy-map component defines the various conditioning actions (metering, marking, shaping, policing, dropping, queuing) that are applied to the classified traffic.
- Application: The service-policy component determines where and in which direction these conditioning policies are enforced.

MQC provides a logical and hierarchical way to implement complex QoS strategies by separating the "what to match" (class-map) from the "what to do" (policy-map) and the "where to apply" (service-policy).

Benefits of using QoS

Benefits of QoS include:

- Control over resources: QoS provides control over bandwidth, which enables prioritization of important traffic and limits less critical traffic, such as FTP transfers, while prioritizing database access.
- Tailored services: Service providers can offer different levels of service to clients by leveraging QoS control and visibility.
- Efficient use of network resources: QoS ensures that mission-critical applications receive the necessary bandwidth and low latency, providing other applications fair service without interference.
- Coexistence of mission-critical applications: QoS ensures that time-sensitive multimedia and voice applications have the bandwidth and minimal delay they require, improving overall network efficiency.
- Foundation for future integrated networks: QoS implementation prepares the network for future multimedia integration. It also supports dynamic QoS signalling protocols like RSVP.

- Improved predictability and performance: QoS provides tools for congestion management, traffic shaping, and policy setting to deliver efficient network services with controlled jitter, latency, and packet loss.
- Support for low latency and bandwidth guarantees: Low latency—minimizing delays in traffic. Bandwidth guarantees—ensuring dedicated bandwidth availability. Traffic policing—controlling traffic rates and managing excess. Queuing mechanisms—organizing traffic for prioritization.

QoS mechanisms

This section provides detailed information about the various mechanisms used by QoS.

Traffic conditioning

Traffic conditioning is a set of mechanisms used in computer networks to manage and control the flow of network traffic, often to ensure Quality of Service (QoS) and meet Service Level Agreements (SLAs). It involves defining bandwidth rules and policies to prioritize, delay, or even drop packets based on specific criteria.

Classification

Classification is the process of identifying and grouping network traffic (packets or flows) based on specified criteria. It is the first and foundational step in implementing QoS on network devices. During classification, a device performs a lookup and assigns a QoS label to a packet. The QoS label identifies all the QoS actions to be performed on a packet and the originating queue of the packet.

Access control lists

Access Control Lists (ACLs) are used to define classes of IP packets that share common characteristics. In QoS configurations, ACLs are not primarily for security (allowing/denying traffic), but rather for identifying traffic flows that require special handling.

Use standard and extended IP ACLs to define a group of packets with the same characteristics (class). IP traffic is classified based on IPv6 ACLs.

ACL types

- Standard ACLs: Match packets based only on source IP address.
- Extended ACLs: Match packets based on source and destination IP addresses, protocol type, source and destination port numbers, etc.
- IPv6 ACLs: Similar to IPv4 ACLs but designed to classify IPv6 traffic using IPv6 addresses and related fields.

ACL actions

- Permit: When a packet matches an ACE with "permit," the specified QoS policy (such as marking or policing) is applied to that packet.
- Deny: Packets matching a "deny" ACE are not selected for QoS actions and are evaluated against the next ACE in the list.

- First-match principle: The packet is compared to the ACEs in order. When the first matching ACE with a "permit" action is found, QoS processing begins for that packet. No further ACLs are evaluated.
- Multiple ACLs on a port: If several ACLs are configured, lookup stops after the first permit match and QoS processing starts, based on that ACL.

Note

- Layer 2-based ACLs are not supported.
- The system does not support deny-based ACL classification; it only supports permit based ACL.

Class maps

A class map is a mechanism that allows you to name a specific traffic flow (or class) and isolate it from all other traffic. The class map specifies the criteria to match and further classify specific traffic flows. The criteria can include matching the access group defined by the ACL, or matching a specific list of DSCP, IP precedence, CoS, or traffic-class values. If you have more than one type of traffic that you want to classify, you can create another class map and use a different name.

To create a class map, use the **class-map** global configuration command. You should use the **class-map** command when the map is shared among multiple policies. When you enter the **class-map** command, the device enters the class-map configuration mode.

You can create a default class by using the **class class-default** policy-map configuration command. The default class is system-defined and cannot be configured. Unclassified traffic, which does not meet the match criteria specified in the traffic classes, is treated as default traffic.

Note

- When configuring a class-map for traffic-class, only one match traffic-class is supported.
- Defining multiple access lists in one class map is not supported.

```
class-map acl1
match access-group name acl1
match access-group name acl2
.............
```

- Applying a class map ACL1 to the policy map is not supported — a class map must have only oneaccess group name, based on either IPv4 or IPv6 packets.

Time-to-live classification

Time-to-Live (TTL) is an 8-bit field in the IPv4 header (called hop limit in IPv6). It specifies the maximum number of hops (routers) a packet can traverse before being discarded. Each router that forwards the packet decreases the TTL by 1. When TTL reaches 0, the packet is dropped.

Note IPv6 TTL is not supported.

TTL classification means identifying and grouping packets based on their TTL value.

This is typically used for:

- Detecting abnormal routing or loops (unexpectedly low TTL values).
- Differentiating packets based on their expected path length or proximity to the source.
- Applying special QoS policies to packets nearing the end of their route (low TTL).

In QoS, network devices can match packets with specific TTL values using ACLs or class maps. Once classified, these packets can be subject to various QoS actions, such as prioritization, policing, or marking.

An example of TTL classification is presented:

```
ip access-list extended IPV4_TTL
permit ip any any ttl eq 100
permit tcp any any ttl ne 150
class-map match-all IPV4_TTL
match access-group name IPV4_TTL
policy-map TTL_MATCH
class IPV4_TTL
police rate 6000000000
set dscp af23
!

Device# show policy-map interface hun1/0/47

HundredGigE1/0/47
Service-policy input: TTL_MATCH
Class-map: IPV4_TTL (match-all)
553567424 packets
Match: access-group name IPV4_TTL
police:
rate 6000000000 bps, burst 187500000 bytes
conformed 22983406600 bytes; actions:
transmit
exceeded 32375773000 bytes; actions:
drop
conformed 588922000 bps, exceeded 830894000 bps
QoS Set
dscp af23
Class-map: class-default (match-any)
2184433710 packets
Match: any
```

Policy maps

Policy maps are configuration objects that form a fundamental component in network QoS configurations, primarily used to define and apply specific QoS policies to network traffic. They act as a container for various QoS actions and are applied to interfaces to control how traffic is handled.

The main purpose of policy maps is to:

- Apply QoS policies: They allow network administrators to define how different types of traffic should be treated, such as prioritizing voice over data, limiting bandwidth for certain applications, or ensuring minimum bandwidth for critical services.
- Modularize QoS configurations: Policy maps provide a structured way to group various QoS mechanisms (like classification, marking, policing, shaping, and queuing) into a single, reusable policy.
- Enforce Service Level Agreements (SLAs): By controlling traffic flow, policy maps help ensure that network performance meets the requirements outlined in SLAs.

Policy map structure

A policy map is typically composed of one or more "class maps." Each class map identifies a specific type of traffic, and within the policy map, actions are defined for that identified traffic class. The general structure is:

```
policy-map [policy-map-name] class [class-map-name]
[QoS action 1]
[QoS action 2]
...
class [class-map-name] [QoS action 1]
[QoS action 2]
...
class class-default [default QoS action]
```

Policy map key components

- Class maps:
  - Before a policy map can be created, traffic must be classified using class maps. A class map defines the criteria for matching specific traffic. These criteria can include:
    - ACLs
    - IP precedence or Differentiated Services Code Point (DSCP) values
    - Protocol (example: TCP, UDP)
    - Port numbers (example: HTTP, HTTPS, SIP)
  - A policy map then references these pre-defined class maps.
- QoS actions: Once traffic is matched by a class map within a policy map, various QoS actions can be applied:
  - Marking: Changing the QoS bits in a packet header (example: DSCP, IP precedence, CoS) to indicate its priority for subsequent network devices.
  - Policing: Dropping or remarking packets that exceed a configured rate. This enforces a strict traffic limit.
  - Shaping: Buffering and delaying excess packets to smooth out traffic bursts and prevent drops, ensuring a consistent output rate.
  - Bandwidth allocation: Reserving a specific amount of bandwidth for a class of traffic.
  - Low Latency Queuing (LLQ): Prioritizing time-sensitive traffic (like voice) to ensure minimal delay and jitter.
  - Weighted Fair Queuing (WFQ)/Class-Based Weighted Fair Queuing (CBWFQ): Distributing available bandwidth fairly or based on configured weights among different traffic classes.
  - Congestion avoidance (example: WRED): Proactively dropping packets to prevent network congestion before it becomes severe.

Policy maps application

Policy maps are typically applied to network interfaces (physical or logical) in either the inbound (ingress) or outbound (egress) direction.

- Ingress policy: Applied to traffic entering an interface.
- Egress policy: Applied to traffic leaving an interface.

By strategically applying policy maps at various points in the network, administrators can implement end-to-end QoS strategies, ensuring optimal performance for critical applications and efficient utilization of network resources.

Default QoS mapping

Tables list default QoS mapping values, applicable if policy map is not applied to an interface.

Table 1: DSCP default mapping table for ingress

DSCP	DSCP	Encapsulation	QoS-group	Traffic-class	DropPrecedence
0	0	0	0	0	G
1	1	0	0	0	G
2	2	0	0	0	G
3	3	0	0	0	G
4	4	0	0	0	G
5	5	0	0	0	G
6	6	0	0	0	G
7	7	0	0	0	G
8	8	1	0	0	G
9	9	1	0	0	G
10	10	1	0	0	G
11	11	1	0	0	G
12	12	1	0	0	G
13	13	1	0	0	G
14	14	1	0	0	G
15	15	1	0	0	G
16	16	2	0	7	G
17	17	2	0	0	G
18	18	2	0	0	G
19	19	2	0	0	G
20	20	2	0	0	G
21	21	2	0	0	G
22	22	2	0	0	G
23	23	2	0	0	G

DSCP	DSCP	Encapsulation	QoS-group	Traffic-class	DropPrecedence
24	24	3	0	7	G
25	25	3	0	0	G
26	26	3	0	0	G
27	27	3	0	0	G
28	28	3	0	0	G
29	29	3	0	0	G
30	30	3	0	0	G
31	31	3	0	0	G
32	32	4	0	7	G
33	33	4	0	0	G
34	34	4	0	0	G
35	35	4	0	0	G
36	36	4	0	0	G
37	37	4	0	0	G
38	38	4	0	0	G
39	39	4	0	0	G
40	40	5	0	7	G
41	41	5	0	0	G
42	42	5	0	0	G
43	43	5	0	0	G
44	44	5	0	0	G
45	45	5	0	0	G
46	46	5	0	7	G
47	47	5	0	0	G
48	48	6	0	7	G
49	49	6	0	0	G
50	50	6	0	0	G
51	51	6	0	0	G
52	52	6	0	0	G
53	53	6	0	0	G
54	54	6	0	0	G
55	55	6	0	0	G

DSCP	DSCP	Encapsulation	QoS-group	Traffic-class	DropPrecedence
56	56	7	0	7	G
57	57	7	0	0	G
58	58	7	0	0	G
59	59	7	0	0	G
60	60	7	0	0	G
61	61	7	0	0	G
62	62	7	0	0	G
63	63	7	0	0	G

Table 2: CoS-drop eligible indicator mapping table for ingress

CoS-DEI	CoS-DEI	Encapsulation	QoS-Group	Traffic-class	Drop precedence
0	0	0	0	0	G
1	1	0	0	0	G
2	2	1	0	0	G
3	3	1	0	0	G
4	4	2	0	0	G
5	5	2	0	0	G
6	6	3	0	0	G
7	7	3	0	0	G
8	8	4	0	7	G
9	9	4	0	7	G
10	10	5	0	7	G
11	11	5	0	7	G
12	12	6	0	0	G
13	13	6	0	0	G
14	14	7	0	0	G
15	15	7	0	0	G

Table 3: MPLS-experimental mapping table for ingress

MPLS-experimental	MPLS-experimental	Encapsulation	QoS-group	Traffic-class	Drop precedence
0	0	0	0	0	G

MPLS-experimental	MPLS-experimental	Encapsulation	QoS-group	Traffic-class	Drop precedence
1	1	1	0	0	G
2	2	2	0	0	G
3	3	3	0	0	G
4	4	4	0	0	G
5	5	5	0	0	G
6	6	6	0	0	G
7	7	7	0	0	G

Table 4: DSCP mapping table for egress

DSCP	DSCP	Encapsulation type of service	Encapsulation priority code point	Encapsulation MPLS-experimental
0	0	0	0	0
1	1	4	0	0
2	2	8	0	0
3	3	12	0	0
4	4	16	0	0
5	5	20	0	0
6	6	24	0	0
7	7	28	0	0
8	8	32	1	1
9	9	36	1	1
10	10	40	1	1
11	11	44	1	1
12	12	48	1	1
13	13	52	1	1
14	14	56	1	1
15	15	60	1	1
16	16	64	2	2
17	17	68	2	2
18	18	72	2	2
19	19	76	2	2
20	20	80	2	2

DSCP	DSCP	Encapsulation type of service	Encapsulation priority code point	Encapsulation MPLS-experimental
21	21	84	2	2
22	22	88	2	2
23	23	92	2	2
24	24	96	3	3
25	25	100	3	3
26	26	104	3	3
27	27	108	3	3
28	28	112	3	3
29	29	116	3	3
30	30	120	3	3
31	31	124	3	3
32	32	128	4	4
33	33	132	4	4
34	34	136	4	4
35	35	140	4	4
36	36	144	4	4
37	37	148	4	4
38	38	152	4	4
39	39	156	4	4
40	40	160	5	5
41	41	164	5	5
42	42	168	5	5
43	43	172	5	5
44	44	176	5	5
45	45	180	5	5
46	46	184	5	5
47	47	188	5	5
48	48	192	6	6
49	49	196	6	6
50	50	200	6	6
51	51	204	6	6

DSCP	DSCP	Encapsulation type of service	Encapsulation priority code point	Encapsulation MPLS-experimental
52	52	208	6	6
53	53	212	6	6
54	54	216	6	6
55	55	220	6	6
56	56	224	7	7
57	57	228	7	7
58	58	232	7	7
59	59	236	7	7
60	60	240	7	7
61	61	244	7	7
62	62	248	7	7
63	63	252	7	7

Table 5: CoS-DEI mapping table for egress

CoS-DEI	CoS-DEI	Encapsulation type of service	Encapsulation priority code point	Encapsulation MPLS-experimental
0	0	0	0	0
1	1	0	0	0
2	2	32	1	1
3	3	32	1	1
4	4	64	2	2
5	5	64	2	2
6	6	96	3	3
7	7	96	3	3
8	8	128	4	4
9	9	128	4	4
10	10	160	5	5
11	11	160	5	5
12	12	192	6	6
13	13	192	6	6
14	14	224	7	7
15	15	224	7	7

Table 6: MPLS-experimental mapping table for egress

MPLS-experimental	MPLS-experimental	Encapsulation type of service	Encapsulation priority code point	Encapsulation MPLS-experimental
0	0	0	0	0
1	0	32	1	1
2	0	64	2	2
3	0	96	3	3
4	0	128	4	4
5	0	160	5	5
6	0	192	6	6
7	0	224	7	7

Metering

Metering is the process of measuring the rate of traffic flow in a network and determining whether the traffic complies with a predefined profile or contract (such as a maximum bandwidth or burst size).

It is a fundamental part of traffic conditioning and Quality of Service (QoS).

How metering works?

Metering checks if packets in a traffic stream conform to specified rate limits (example, X Mbps with a certain burst size).

Each packet (or burst of packets) is "colored" or categorized (typically as conformant, exceeding, or violating) based on whether it fits within the contract.

Typical metering results

- Conformant (Green): Packet fits within the allowed rate and burst; eligible for normal forwarding.
- Exceeding (Yellow): Packet exceeds the committed rate but is within allowable burst; may be forwarded with reduced priority or re-marked.
- Violating (Red): Packet exceeds both the rate and burst; typically dropped or severely de-prioritized.

Where is metering used?

- As part of policing (to enforce compliance with traffic profiles).
- As part of shaping (to smooth traffic to a desired rate).
- Metering on Port-channel subinterface is not supported.

Marking

Marking is the process of setting specific fields in packet headers to indicate the traffic’s class or priority. This allows network devices along the path to identify and provide differentiated treatment (such as prioritization, shaping, or dropping) based on these markings.

Marking is used on traffic to convey specific information to a downstream device in the network, or to carry information from one interface in a device to another. When traffic is marked, QoS operations on that traffic can be applied. This can be accomplished directly using the **set** command or through a table map, which takes input values and translates them directly to values in the output.

Marking can be used to set certain field/bits in the packet headers, or marking can also be used to set certain fields in the packet structure that is internal to the device. It can also be used with traffic classes for packets to hit specific VoQs. Additionally, the marking feature can be used to define mapping between fields.

The following marking methods are available for QoS:

- Packet header
- Device specific information
- Table maps

Packet header marking

Marking on fields in the packet header can be classified into two general categories:

- IPv4/v6 header bit marking
- Layer 2 header bit marking



Note VLAN id based and packet length based classification is not supported.

The marking feature at the IP level is used to set the precedence or the DSCP in the IP header to a specific value to get a specific per-hop behavior at the downstream device (switch or router), or it can also be used to aggregate traffic from different input interfaces into a single class in the output interface. The functionality is currently supported on both the IPv4 and IPv6 headers.

Marking in the Layer 2 headers is typically used to influence dropping behavior in the downstream devices (switch or router). It works in tandem with the match on the Layer 2 headers. The bits in the Layer 2 header that can be set using a policy map are class of service.



Note Based on whether the packet is Layer 2 or Layer 3, DSCP or CoS bit is marked. On a routed traffic, only DSCP marking is allowed, and on a bridged traffic, only CoS marking is allowed.

Switch-specific information marking

This form of marking includes marking of fields in the packet data structure that are not part of the packets header, so that the marking can be used later in the data path. This is not propagated between the switches. Marking of QoS group falls into this category. This form of marking is only supported in policies that are enabled on the input interfaces. The corresponding matching mechanism can be enabled on the output interfaces on the same switch and an appropriate QoS action can be applied.

Traffic-class can be set in the ingress policy to direct the matched traffic into a corresponding VoQ. Discard-class can be set in the ingress policy to be used later for VoQ congestion management for selecting the corresponding queue-limit or random-detect settings based on discard-class.

Table map marking

Table map marking enables the mapping and conversion from one field to another using a conversion table. This conversion table is called a table map.

Depending upon the table map attached to an interface, CoS, DSCP, and Precedence values of the packet are rewritten. The device allows configuring both ingress table map policies and egress table map policies. Table maps can also be used to map or set the incoming traffic to a particular VoQ using traffic-class values (0 to 7). CoS or DSCP or PREC to discard-class on the ingress and QoS group to CoS or DSCP or PREC on the egress are also supported.

Only table-map values of same type of QoS tags are supported. For example, table-map of type CoS to CoS or DSCP to DSCP or PREC to PREC is supported, and also table-map of type PREC to QoS group and DSCP or CoS or PREC to traffic-class is supported.

A table map-based policy supports the following capabilities:

- Mutation: You can have a table map that maps from one DSCP value set to another DSCP value set, and this can be attached to both ingress and egress ports.
- Rewrite: Packets coming in are rewritten depending upon the configured table map.
- Mapping: Table map based policies can be used instead of set policies.

The following steps are required for table map marking:

- Define the table map: Use the **table-map** global configuration command to map the values. The table does not know of the policies or classes within which it will be used. The default command in the table map is used to indicate the value to be copied into the field when there is no matching from field.
- Define the policy map: You must define the policy map where the table map will be used.
- Associate the policy to an interface.



Note

- DSCP to CoS Table Maps are not supported. You cannot use table maps to translate DSCP values directly to CoS values.
- QoS Group to DSCP Table Map: When you map a QoS group to a DSCP value (for DSCP marking at egress), this does not set the QoS group value in the packet.
- If you need the egress packet to have a specific (non-zero) CoS value, configure a separate QoS group to CoS table map.

Policing

The QoS policing feature is used to impose a maximum rate on a traffic class. If the rate is exceeded, then a specific action is taken as soon as the event occurs.

The rate parameters (committed information rate [CIR] and peak information rate [PIR]) are configured in bits per second, and the burst parameters (conformed burst size [Bc] and extended burst size [Be]) are configured in bytes per second.

Only the single-rate two-color policing forms are supported for QoS.

Policing uses a token-bucket algorithm. As each frame is received by the device, a token is added to the bucket. The bucket has a hole in it and leaks at a rate that you specify as the average traffic rate in bits per second. Each time a token is added to the bucket, the device verifies that there is enough room in the bucket. If there is not enough room, the packet is marked as nonconforming, and the specified policer action is taken (dropped or marked down).

How quickly the bucket fills is a function of the bucket depth (burst-byte), the rate at which the tokens are removed (rate-bps), and the duration of the burst above the average rate. The size of the bucket imposes an upper limit on the burst length and limits the number of frames that can be transmitted back-to-back. If the burst is short, the bucket does not overflow, and no action is taken against the traffic flow. However, if a burst is long and at a higher rate, the bucket overflows, and the policing actions are taken against the frames in that burst.

Single-rate two-color policing

- Configuration:
  - Only the Committed Information Rate (CIR) is specified.
  - Committed Burst Size (Bc) is optional; if not specified, it defaults to a calculated value.
- Operation:
  - Uses a single token bucket.
  - If an incoming packet finds enough tokens, it is conforming (within CIR).
  - If not, it is non-conforming (exceeds CIR) and may be dropped or remarked.

Shaping

Shaping is the process of imposing a maximum rate of traffic while regulating the traffic rate in such a way that downstream devices are not subjected to congestion. Shaping in the most common form is used to limit the traffic sent from a physical or logical interface.

Shaping has a buffer associated with it that ensures that packets which do not have enough tokens are buffered as opposed to being immediately dropped. The number of buffers available to the subset of traffic being shaped is limited and is computed based on a variety of factors. The number of buffers available can also be tuned using specific QoS commands.

Packets are buffered as buffers are available, beyond which they are dropped.

Class-based traffic shaping

The device uses class-based traffic shaping. This shaping feature is enabled on a class in a policy that is associated to an interface. A class that has shaping configured is allocated a number of buffers to hold the packets that do not have tokens. The buffered packets are sent out from the class using FIFO. In the most common form of usage, class-based shaping is used to impose a maximum rate for a physical interface or logical interface as a whole.

Shaping is implemented using a token bucket. The values of CIR, Bc and Be determine the rate at which the packets are sent out and the rate at which the tokens are replenished.

Average rate traffic shaping

You use the **shape average** policy-map class command to configure average rate shaping. This command configures a maximum bandwidth for a particular class. The queue bandwidth is restricted to this value even though the port has more bandwidth available. The device supports configuring shape average by either a percentage or by a target bit rate value.

Queuing and scheduling

Queueing is the process of placing packets into buffers (queues) when they cannot be transmitted immediately due to congestion or bandwidth limitations. Different types of traffic can be assigned to different queues based on their priority or classification.

Scheduling is the method used to decide the order and rate at which packets are removed from queues and transmitted.

Queuing and scheduling are both used to prevent traffic congestion. Traffic is sent to specific queues for servicing and scheduling, based upon bandwidth allocation. Traffic is then scheduled or sent out through the port. Traffic class is used as the classification value for queueing.

The device supports the following queuing and scheduling features:

- Bandwidth remaining ratio
- Priority queues
- Queue buffers
- Weighted Random Early Detection

Bandwidth

In QoS, bandwidth refers to the amount of network capacity that is reserved or guaranteed for a specific class of traffic.

Allocating bandwidth ensures that critical applications receive the minimum resources they need, even during network congestion.

Bandwidth allocation determines the available capacity for traffic that is subject to QoS policies.

How is bandwidth used in QoS?

- Bandwidth guarantees: You can assign a guaranteed minimum bandwidth to different traffic classes (example: voice, video, data) to ensure service quality.
- Bandwidth limits: You can also cap the maximum bandwidth for certain classes, preventing them from consuming all available resources.
- Bandwidth sharing: When not all bandwidth is used by one class, excess can often be shared by other classes (depending on configuration).

Priority queues

A priority queue is a special type of traffic queue in network devices (such as routers and switches) that always gets serviced before other queues.

It ensures that delay-sensitive traffic—such as voice or real-time video—is sent with the lowest possible latency and jitter.

How priority queues work?

- Strict priority: The scheduler always services the priority queue first, forwarding all packets in this queue before attending to other queues.
- Bandwidth limit: In modern QoS designs (like Cisco's Low Latency Queuing), the priority queue can be assigned a maximum bandwidth to prevent it from starving other traffic classes.

How scheduling works with priority queues?

- The priority queue is checked first. If there is any traffic, it is sent immediately.
- Only when the priority queue is empty does the scheduler send packets from other queues.
- If the priority queue has a bandwidth cap, the CLI is not supported when a class is defined with apriority queue.

Priority levels on Cisco C9350 series smart switches

Each port supports eight ingress queues, of which seven can be given a priority.

You can use the **priority level** policy class-map command to configure the priority of two classes. One is the class-default which is the lowest priority level, and the other priority level 1 which is also traffic class 7. Each priority level can be configured to one class only, that is, there cannot be more than one class at the same priority level. Each class that does not have a priority level configured is referred to as priority normal, and there can be multiple priority normal classes.

If a priority level is configured in a policy-map, priority level 1 must be configured. For example, policy-map with priority level 2 and priority level 3 but no priority level 1 is not allowed. If all the priority levels configured in a policy-map is sorted, they must be contiguous, that is, no priority level should be skipped, for example, the sequence of priority levels 1, 2, 4, and 5 where priority level 3 is skipped is not allowed.

Each priority level must be configured to the class that matches the corresponding traffic class as shown in this table:

Table 7: Priority level to traffic class mapping

Priority level	Traffic class
1 (highest)	7
2	6
3	5
4	4
5	3
6	2
7	1

Priority level	Traffic class
None (lowest)	0

Priority levels on Cisco C9610 series smart switches

You can use the priority level policy class-map command to configure the priority of two classes. Traffic class 7 should always be configured with priority level 1, and traffic class 6 can be either priority level 2 or non-priority. Priority level 2 cannot be used with any other traffic class.

Queue buffers

Queue buffers are memory spaces in network devices (such as routers and switches) where packets are temporarily stored (queued) when they cannot be transmitted immediately due to congestion or bandwidth limitations.

Why are queue buffers important?

- Absorb bursts: They help absorb temporary bursts of traffic, preventing immediate packet loss when the outgoing link is busy.
- Enable queueing policies: Allow for advanced queueing and scheduling mechanisms (example: priority queueing, fair queueing).
- Protect critical traffic: When combined with QoS classification, queue buffers can ensure high-priority traffic is held and sent first.

How queue buffers work?

- Packet arrival: Packets arrive faster than they can be transmitted.
- Buffering: Excess packets are placed in queue buffers instead of being dropped right away.
- Scheduling: Packets are removed from buffers and sent according to the scheduling policy (example: priority, round-robin).

Weighted random early detection

Weighted Random Early Detection (WRED), commonly known as Weighted Tail Drop, is a congestion avoidance mechanism used in Cisco devices to manage queue lengths and prevent buffer overflow by selectively dropping packets before the queue becomes full. Unlike traditional tail drop, which drops packets only when the queue is completely full, WRED proactively drops packets based on the average queue size and packet priority, helping to avoid global synchronization and improve overall network performance.

How WRED works?

- Queue monitoring: WRED monitors the average queue length for each traffic class.
- Thresholds and probabilities:
  - Minimum threshold: Below this, packets are not dropped.
  - Maximum threshold: Above this, all packets are dropped (tail drop).

  - Between thresholds: Packets are dropped randomly, with a probability that increases as the queue length grows.

- Weighted by priority: Higher-priority traffic (higher DSCP or precedence values) can be assigned higher thresholds, meaning they are less likely to be dropped under congestion.

Benefits of WRED

- Prevents global TCP synchronization: Early random drops prompt TCP flows to reduce their rates at different times, improving overall throughput.
- Differentiated drop behavior: Protects high-priority traffic by making it less likely to be dropped.
- Reduces buffer overflows and latency spikes compared to traditional tail drop.

Trust behavior

Trust behavior refers to the way a network device processes QoS markings, like DSCP, IP precedence, or CoS, in packets received on an interface. Trusting means relying on received QoS markings. Not trusting means classifying and marking according to local policy.

Correct trust configuration ensures QoS that is effective, fair, and secure across the network.

- To "trust" a field: The device utilizes existing QoS marking in the packet (as set by upstream devices).
- To "not trust" a field: The device ignores or overwrites the existing marking, typically re-classifying and re-marking the packet based on its own policies.

Why is trust behavior important?

- Consistency: Ensures consistent QoS policy enforcement across the network.
- Security: Prevents users or devices from abusing QoS by marking all their traffic as high-priority.
- Interoperability: Helps networks interconnect without conflicting QoS policies.

Where is trust behavior configured?

Trust is usually configured on ingress (incoming) interfaces—especially at the edge of a network (access ports, WAN connections, or interconnection points).

Common trust options

On Cisco switches and routers, you can typically trust:

- Differentiated Services Code Point (DSCP)
- IP precedence
- Class of Service (CoS, in Ethernet 802.1Q/802.1p environments)
- None (do not trust; reclassify/re-mark)

How QoS works

Typically, networks operate on a best-effort delivery basis, giving all traffic equal priority and an equal chance for timely delivery. When congestion occurs, all traffic has an equal chance of being dropped.

The QoS mechanism consists of three basic steps:

- Classify traffic types.
- Specify actions against them.
- Determine where actions should occur.

When you configure the QoS feature, you should select specific network traffic, prioritize it based on its importance, and apply congestion-management and avoidance techniques to ensure preferential treatment. Implementing QoS in your network makes performance predictable and utilization effective.

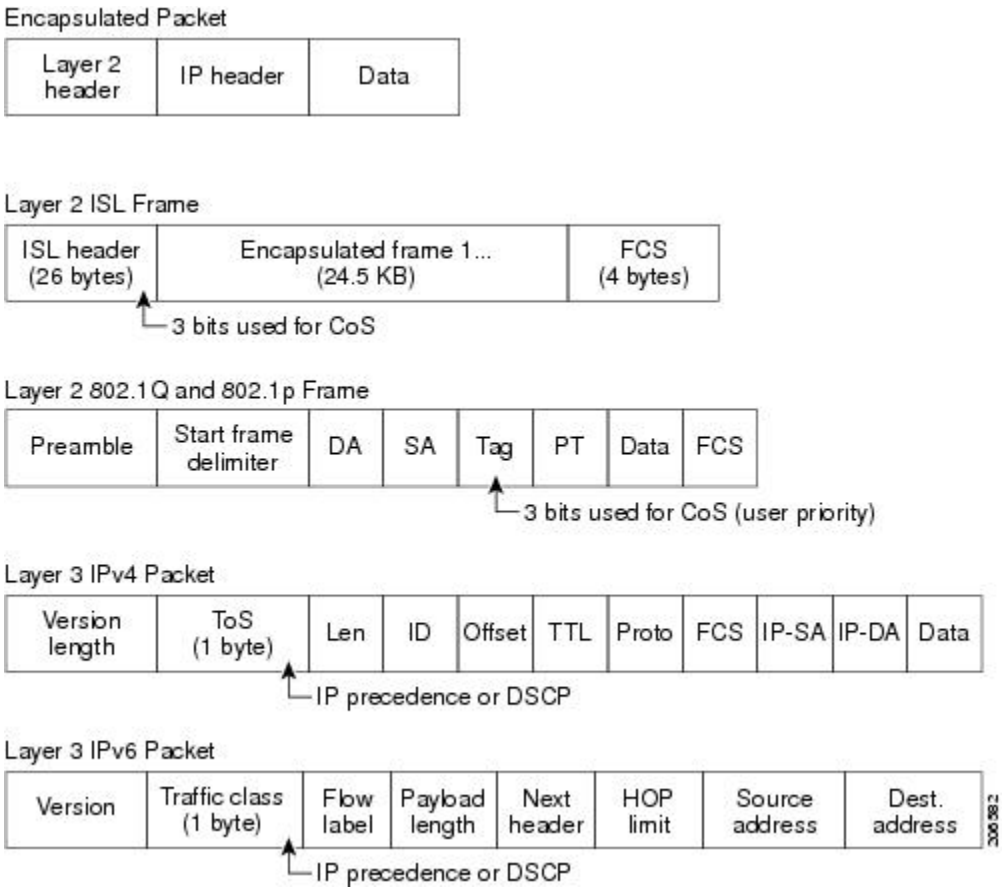
Differentiated Services architecture

QoS implementation is based on the Differentiated Services (Diff-Serv) architecture, a standard from the Internet Engineering Task Force (IETF). This architecture specifies that each packet is classified upon entry into the network.

The classification is carried in the IP packet header using six bits from the deprecated IP Type of Service (ToS) field to carry the classification (class) information. Classification can also be carried in the Layer 2 frame.

The special bits in the Layer 2 frame or a Layer 3 packet are shown in this figure:

Figure 1: Classification layers of QoS in frames and packets



Layer 3 packet prioritization bits

Layer 3 IP packets can carry an IP Precedence value or a DSCP value. QoS supports the use of either value because DSCP values are backward-compatible with IP precedence values.

IP precedence values range from 0 to 7. DSCP values range from 0 to 63.

How QoS uses classification

All switches and routers accessing the internet rely on class information. A switch provides the same forwarding treatment to packets with identical class information and different treatment to packets with varying class information. The class information in the packet can be assigned by end hosts or by switches or routers along the way, based on a configured policy, detailed examination of the packet, or both. It is expected that detailed examination of the packet will occur closer to the edge of the network, avoiding overload for the core switches and routers.

Switches and routers along the path can use the class information to limit the number of resources allocated per traffic class. The behavior of an individual device when handling traffic in the Diff-Serv architecture is called per-hop behavior. If all the devices along a path provide a consistent per-hop behavior, you can construct a complete QoS solution.

Implementing QoS in your network can be simple or complex, depending on the features offered by your internetworking devices, the types and patterns of traffic in your network, and the level of control you need over incoming and outgoing traffic.

Packet classification

Packet classification is the process of identifying a packet as belonging to one of several classes in a defined policy, based on certain criteria. The MQC is a policy-class based language.

The policy class language is used to define these:

- Class map template with one or several match criteria
- Policy map template with one or several classes associated to the policy map

The policy map template is then associated to one or several interfaces on the device. Packet classification is the process of identifying a packet as belonging to one of the classes defined in the policy map. The process of classification will exit when the packet being processed matches a specific filter in a class. This is referred to as first-match exit. If a packet matches multiple classes in a policy, irrespective of the order of classes in the policy map, it will still exit the classification process after matching the first class.

If a packet does not match any of the classes in the policy, it is classified into the default class in the policy. Every policy map has a default class, which is a system-defined class to match the packets that do not match any of the user-defined classes.

Packet classification can be categorized into these types:

- Classification based on information that is propagated with the packet
- Classification based on information that is device specific
- Hierarchical classification (on Cisco C9350 series smart switches)

Classification based on packet propagated information

Classification based on information that is part of a packet, and propagated either end-to-end or between hops, typically includes these:

- Classification based on Layer 3 or 4 headers
- Classification based on Layer 2 information

Classification based on Layer 3 or Layer 4 header

This is the most common deployment scenario. Numerous fields in the Layer 3 and Layer 4 headers can be used for packet classification.

At the most granular level, this classification methodology can be used to match an entire flow. For this deployment type, an access control list (ACL) can be used. ACLs can also be used based on various subsets of the flow, for example, source IP address only, destination IP address only, or a combination of both.

Classification can also be done based on the precedence or DSCP values in the IP header. The IP precedence field is used to indicate the relative priority with which a particular packet needs to be handled. It is made up of three bits in the IP header's type of service (ToS) byte.

This table shows the different IP precedence bit values and their names.

Table 8: IP precedence values and names

IP precedence value	IP precedence bits	IP precedence names
0	000	Routine
1	001	Priority
2	010	Immediate
3	011	Flash
4	100	Flash override
5	101	Critical
6	110	Internetwork control
7	111	Network control

Note All the routing control traffic in a network uses the IP precedence value 6 by default. IP precedence value 7 is also reserved for network control traffic. Therefore, we do not recommend the use of IP precedence values 6 and 7 for user traffic.

The DSCP field is made up of 6 bits in the IP header and is being standardized by the IETF Differentiated Services working group. The original ToS byte, which contained the DSCP bits, has been renamed the DSCP byte.

The DSCP field is part of the IP header, similar to IP precedence. The DSCP field is a super set of the IP precedence field. Therefore, the DSCP field is used and is set in ways that are similar to what was described with respect to IP precedence.

Note The DSCP field definition is backward-compatible with the IP precedence values. Some fields in the Layer 2 header can also be set using a policy.

Classification based on Layer 2 header

The CoS method can be used to perform classification based on the Layer 2 header information. Classification is based on the three bits in the Layer 2 header based on the IEEE 802.1p standard. This usually maps to the ToS byte in the IP header.

Classification based on device specific information

A device also provides classification mechanisms in scenarios that are available where classification is not based on information in the packet header or payload.

At times you might be required to aggregate traffic coming from multiple input interfaces into a specific class in the output interface. For example, multiple customer edge routers might be going into the same access device on different interfaces. The service provider might want to police all the aggregate voice traffic going into the core to a specific rate. However, the voice traffic coming in from the different customers could have different ToS settings. QoS group-based classification is a feature that is useful in these scenarios.

Policies configured on the input interfaces set the QoS group to a specific value, which can then be used to classify the packets in the policies enabled on the output interface.

The QoS group is a field in the packet data structure that is internal to a device. It is important to note that a QoS group is an internal label to the device and is not a part of the packet header.

QoS prerequisites

To implement QoS, the device must perform these tasks:

- Traffic classification: Distinguish packets or flows from one another.
- Traffic marking and policing: Assign a label to indicate the given quality of service as the packets move through the device, and then make the packets comply with the configured resource usage limits.
- Queuing and scheduling: Provide various treatments where resource contention exists.
- Shaping: Ensure the sent traffic meets a specific profile.
- Ingress port activity: Activities that occur at the ingress port of a device:
  - Classification: Classifying a distinct path for a packet by associating it with a QoS label. For instance, the device maps the CoS or DSCP in the packet to a QoS label to distinguish different types of traffic. The QoS label that is generated identifies all future QoS actions to be performed on this packet.
  - Marking: Marking evaluates the policer and decides what to do with a packet: pass it through without modification, mark down the QoS label, or drop it. Traffic can be marked with traffic-class to hit a particular VoQ. Marking can also be done without policing.
- Egress ort activity: Activities that occur at the egress port of a device:
  - Policing: Policing is not supported for outgoing packets.
  - Marking: Marking rewrites the QoS tags such as DSCP and CoS in the packet so that the next hop can apply QoS based on the new value. You must configure a separate policy-map and use the **service-policy output policy-name** command to attach it to the egress direction of an interface. It is different from the type queueing policy-map described in the next bullet point. This policy-map can only contain marking action using the **set qos-tag value** command. For example, set dscp cs6 or set cos 5.
  - Queuing: Queuing selects which VoQ to use and applies congestion management such as queue-limit and random-detect. Packets admitted into VoQ are scheduled to the corresponding egress output queueing. Shape, priority, and bandwidth remaining ratio affects how packets are scheduled. VoQ selection happens on the ingress side by setting the traffic-class in the ingress policy if configured, or it follows the default QoS tag to traffic-class mapping which only uses traffic-class 0 and 7. On the egress side, a separate **type queueing** policy-map can be attached to the interface unless the default queuing configuration is acceptable. This policy-map can only use class-maps that match traffic-class (class-default matches traffic-class 0 by default). The **service-policy type queueing output policy-name** command is used to attach the queuing policy-map to an interface. The policy-name used must be a policy-name defined using the **policy-map type queueing** command.

QoS limitations

This section lists the limitations of QoS feature.

Limitations on wired targets

- Queueing:
  - Max 8 queuing classes per port for a wired target.
  - Only 8 output queues per port supported.
  - Queue-limit range varies between SUP-3 and SUP-3XL.
- Policing (non-queueing policy map):
  - Policer supported only on ingress.
  - Upto 32 policers per ingress policy map per port.
  - Each policer tracks conform, exceed, and violate counters.
- Policy maps:
  - Max 1599 policy maps and 16 unique policy maps per device.
  - Max 256 classes per policy on a wired port.
- Priority levels on Cisco C9350 series smart switches:
  - 2 priority levels supported.
  - Only one priority level per class map; default is 0 (nonpriority).
- Priority levels on Cisco C9610 series smart switches:
  - Only 2 priority levels supported.
  - Only one priority level per class map; default is 0 (nonpriority).
  - Traffic class 7 should always be configured with priority level 1, and traffic class 6 can be either priority level 2 or non-priority. Priority level 2 cannot be used with any other traffic class.
  - Multicast traffic can be shaped on priority classes. Shaping does not take any impact on non-priority queues for multicast traffic.
- Hierarchical QoS (HQoS): Applicable for Cisco C9350 series smart switches and Cisco C9550 series smart switches.
  - No overlapping actions between parent and child (except port shaper in parent and queuing in child).
  - Policing or marking can only be configured in parent or child, not both.
  - For ingress HQoS, only parent-level policing is supported.
  - Empty classes are allowed.

- Marking:
  - Not supported in egress type queuing policy.
  - Egress remarking requires a new policy map (DSCP, PREC, CoS, MPLS EXP, or QoS-group based only, not ACL).
  - No DSCP to CoS, CoS to precedence, or similar table maps (must be of the same QoS type).
- Classification counters:
  - Port-based only; no aggregation.
  - Packets, not bytes are counted.
  - Triggered only by marking or policing actions.
  - Ingress: 32 unique counters/port
  - Egress: 8 unique counters/port; extra classes share counters.
  - Counter is cumulative for all matches in a class.
- Policing:
  - Not supported in egress direction.
  - Not supported on SVI, tunnel interfaces, and member add port channel sub interface.
  - Aggregate policing for egress not supported.
- Queueing:
  - Supported only on Traffic Class (TC) based classification.
  - Not supported on SVI, port channel interface, and tunnel interface.
  - No Application Visibility and Control (AVC) or NBAR based QoS.
- Auto QoS:
  - Auto QoS is supported on Layer 2 and Layer 3 physical ports.
  - Auto QoS is not supported on port-channel member ports, port-channel interfaces,subinterfaces, SVIs, and tunnel interfaces.
- Table maps:
  - Only one table-map for exceed, one for violate markdown per switch/stack.
  - Max 15 unique combinations of exceed/violate table maps supported.
  - All classes in a policy map must use the same table-map for exceed/violate.
- Egress remarking:
  - Not supported with ACL-based classification.
  - Only supported via DSCP, CoS, precedence, or MPLS EXP (new policy map required).

- GRE tunnels:
  - Policy map can only be attached to physical members, not logical tunnel interfaces.
  - GRE policing and marking not supported.
- LISP QoS:
  - Not supported.
- QoS ACL TCAM for egress:
  - Not supported.
- BUM (Broadcast, Unknown Unicast, Multicast) traffic:
  - Limited visibility in show policy-map type queue interface.
  - Only 2 output queue stats for multicast.
- Classification not supported for:
  - VPLS
  - Layer 2/MAC VLAN ID
  - Protocol types
  - Packet length (fixed/range)
  - RTP header/type
  - Egress side ACLs
  - IPv6 ACE with TCP flag
  - Object group-based ACLs
  - App-ID metadata
  - Security Group Tag (SGT) aware QoS
  - StackWise Virtual Link (SVL) QoS

Limitations on EtherChannel and channel member interfaces

- Queuing policy:
  - Supported only on EtherChannel member ports.
  - You can apply a queuing-type service policy to individual member ports of an EtherChannel.
- Non-queuing policy:
  - Supported only on the EtherChannel interface itself.

- Non-queuing (example: policing, marking) policies must be applied to the Port-Channel (logical) interface, not to the individual member ports.

- Policy rejection:
  - If you attempt to apply a non-queuing service policy to a channel member port, the configuration is rejected with a message:

```
Only queueing type policy is supported on member interfaces.
```

Limitations on egress queuing policy

- Single traffic class per class-map: Each class-map in a queuing policy can only match one traffic class (range: 0–7).
- Class-default: Always matches traffic class 0. No other class-map can match traffic class 0.
- No set traffic-class: If an ingress policy map does not include a set traffic-class action, the QoS tag is mapped to traffic class 0 or 7 by default.
- Override system default: To use non-default queues, configure the set traffic-class action in the ingress policy map. This setting influences Virtual Output Queue (VOQ) selection for all egress ports.
- Profile limit: Each unique combination of traffic classes in a queuing policy-map requires its own profile.
- Max Profiles: Only eight traffic class profiles are supported for main interfaces.
- Packet drop stats:
  - Packet drop statistics for multicast and Layer 2 traffic dropped due to queuing policer (shape limit exceeded) are not displayed in total packet drop stats on the interface.
  - Only unicast drop statistics are shown.
- Priority mapping: Each configured priority level must be set in the class that matches the corresponding traffic class.

Supported actions in queuing policy

Only these actions are supported within a queuing policy map:

- priority
- shape
- bandwidth remaining ratio
- queue-limit
- random early detection (WRED)

Configure Class, Policy, and Maps

This section provides configuration information about class, policy, and maps.

Create a traffic class

Before you begin

All match commands specified in this configuration task are considered optional, but you must configure at least one match criterion for a class.

Note Define individual class-maps for different types of QoS tag values. You cannot match a traffic-class with two different traffic-class values in the same class-map. For example, you cannot add DSCP, PRE, or CoS matching to a class-map that already includes a traffic-class match.

To create a traffic class containing match criteria, use the **class-map** command to specify the traffic class name, and then use the following **match** commands in class-map configuration mode, as needed.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**class-map** *class-map name* {**match-any** | **match-all**}
	Example:
	`Device(config)# class-map test_1000`
	`Device(config-cmap)#`
		Enters class map configuration mode.
		• Creates a class map to be used for matching packets to the class whose name you specify.
		• **match-any**: Any one of the match criteria must be met for traffic entering the traffic class to be classified as part of it.
		• **match-all**: All of the match criteria must be met for traffic entering the traffic class to be classified as part of the traffic class.
		Note
		This is the default. If **match-any** or **match-all** is not explicitly defined, **match-all** is chosen by default.

	Command or Action	Purpose

Step 3	**match access-group** *name*
	Example:
	`Device(config-cmap)# match access-group 100`
	`Device(config-cmap)#`
		The following parameters are available for this command:
		• access-group
		• cos
		• dscp
		• ip
		• mpls
		• precedence
		• qos-group
		• traffic-class

Step 4	**match cos** *cos value*
	Example:
	`Device(config-cmap)# match cos 2 3 4 5`
	`Device(config-cmap)#`
		(Optional) Matches IEEE 802.1Q or ISL class of service (user) priority values.
		• Enters up to 4 CoS values separated by spaces (0 to 7).

Step 5	**match dscp** *dscp value*
	Example:
	`Device(config-cmap)# match dscp af11 af12`
	`Device(config-cmap)#`
		(Optional) Matches the DSCP values in IPv4 and IPv6 packets.

Step 6	**match ip** {**dscp** *dscp value* | **precedence** *precedence value*}
	Example:
	`Device(config-cmap)# match ip dscp af11 af12`
	`Device(config-cmap)#`
		(Optional) Matches IP values including the following:
		• **dscp**: Matches IP DSCP (DiffServ codepoints).
		• **precedence**: Matches IP precedence (0 to 7).
		Note Because CPU-generated packets are not marked at egress, these packets do not match the configured class-map.

Step 7	**match mpls experimental** *experimental value* **Example:** `Device(config-cmap)# match mpls experimental topmost 3`	(Optional) Match MPLS experimental valueon top most label (from 0 to 7).

	Command or Action	Purpose
	`Device(config-cmap)#`	

Step 8	**match qos-group** *qos group value*
	Example:
	`Device(config-cmap)# match qos-group 10`
	`Device(config-cmap)#`
		(Optional) Matches QoS group value (from 0 to 31).

Step 9	**match traffic-class** *traffic-class value*
	Example:
	`Device(config-cmap)# match traffic-class 7`
	`Device(config-cmap)#`
		(Optional) Matches QoS traffic class value (from 1 to 7).

Step 10	**end** **Example:** `Device(config-cmap)# end`	Saves the configuration changes.

What to do next

Configure the policy map.

Create a traffic policy

Before you begin

You should have first created a class map.

To create a traffic policy, use the **policy-map** global configuration command to specify the traffic policy name.

The traffic class is associated with the traffic policy when the **class** command is used. The **class** command must be entered after you enter the policy map configuration mode. After entering the **class** command, the device is automatically in policy map class configuration mode, which is where the QoS policies for the traffic policy are defined.

The following policy map class-actions are supported:

- exit: Exits from the QoS class action configuration mode.
- no: Negates or sets default values for the command.
- police: Policer configuration options.
- service-policy: Configures the QoS service policy.
- set: Sets QoS values using the following options:

- CoS values
- Discard-class values
- DSCP values
- IP values
- MPLS exp values
- Precedence values
- QoS group values
- Traffic class values

Note If the **set** option is not used in the egress policy to remark egress traffic, the statistics of the policy will not be displayed when you use the **show policy-map interface** command.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map** *policy-map name*
	Example:
	`Device(config)# policy-map test_2000`
	`Device(config-pmap)#`
		Enters policy map configuration mode.
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy.

Step 3	**class** {*class-name* | **class-default**}
	Example:
	`Device(config-pmap)# class test_1000`
	`Device(config-pmap-c)#`
		Specifies the name of the class whose policy you want to create or change.
		You can also create a system default class for unclassified packets.

Step 4	**police** {*target_bit_rate* | **cir** | **rate**}
	Example:
	`Device(config-pmap-c)# police 10000000`
	`Device(config-pmap-c)#`
		(Optional) Configures the policer:
		• *target_bit_rate*: Enter the bit rate per second, enter a value between 1200000 and 400000000000.
		• **cir**: Committed Information Rate.

	Command or Action	Purpose
		• **rate**: Specify police rate, PCR for hierarchical policies or SCR for single-level ATM 4.0 policer policies.

Step 5	**service-policy** *policy-map name*
	Example:
	`Device(config-pmap-c)# service-policy input`
	`Device(config-pmap-c)#`
	Note
	Overlapping actions are not supported.
		(Optional) Configures the QoS service policy.

Step 6	**set** {**cos** | **discard-class** | **dscp** | **ip** | **mpls** | **precedence** | **qos-group** | **traffic-class**}
	Example:
	`Device(config-pmap-c)# set cos 7`
	`Device(config-pmap-c)#`
		(Optional) Sets the QoS values. Possible QoS configuration values include:
		• **cos**: Sets the IEEE 802.1Q/ISL class of service/user priority.
		• **discard-class**: Sets discard behavior identifier.
		• **dscp**: Sets DSCP in IP(v4) and IPv6 packets.
		• **ip**: Sets IP specific values.
		• **mpls**: Sets MPLS specific values.
		• **precedence**: Sets precedence in IP(v4) and IPv6 packet.
		• **qos-group**: Sets the QoS Group.
		• **traffic-class**: Sets the traffic class.

Step 7	**end**
	Example:
	`Device(config-pmap-c) #end`
	`Device(config-pmap-c) #`
		Saves the configuration changes.

What to do next

Configure the interface.

Create a traffic queueing policy

To create a traffic queueing policy for QoS, perform this procedure.

The following queueing policy map class-actions are supported:

- bandwidth: Bandwidth configuration options.
- exit: Exits from the QoS class action configuration mode.
- no: Negates or sets default values for the command.
- priority: Configures strict scheduling priority for the class.
- queue-limit: Queue threshold for tail drop configuration options.
- random-detect: Enables Random Early Detection as drop policy.
- service-policy: Configures QoS service policy.
- shape: Traffic shaping configuration options.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map type queueing** *policy name*
	Example:
	`Device(config)# policy-map type queueing test`
	`Device(config-pmap)#`
		Enters policy map configuration mode.
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy.

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class traffic-class7`
	`Device(config-pmap-c)#`
		Enters policy class map configuration mode. Specifies the name of the class whose policy you want to create or change. Command options for policy class map configuration mode include the following:
		• *word*: Class map name.
		• **class-default**: System default class matching any otherwise unclassified packets.

Step 4	**bandwidth remaining ratio** *ratio*
	Example:
	`Device(config-pmap-c)# bandwidth remaining ratio 10`
	`Device(config-pmap-c)#`
		(Optional) Configures ratio for sharing excess bandwidth. The range is from 1 to 63.
		Note
		• **bandwidth remaining ratio** and **priority level** cannot be configured simultaneously for the same class, even though they can be in the same policy.

	Command or Action	Purpose
		• Traffic class 7 must always be configured with priority level 1.

Step 5	**priority level** *level*
	Example:
	`Device(config-pmap-c)# priority level 1`
	`Device(config-pmap-c)#`
		(Optional) Configures priority level queue.

Step 6	**queue-limit** *value* [**bytes**]
	Example:
	`Device(config-pmap-c)# queue-limit 100000 bytes`
	`Device(config-pmap-c)#`
		(Optional) Sets the queue limit threshold percentage value in bytes.

Step 7	**random-detect** {**discard-class** | **discard-class-based** | **exponential-weighting-constant**}
	Example:
	`Device(config-pmap-c)# random-detect discard-class-based`
	`Device(config-pmap-c)#`
		(Optional) Enables Random Early Detection as the drop policy.
		• **discard-class**: Parameters for each discard- class value (0 or 1).
		Note **discard-class** and **priority** cannnot be configured simultaneously for the same class, even though they can be in the same policy.
		• **discard-class-based**: Enables discard-class-based WRED as the drop policy.
		• **exponential-weighting-constant**: Weight for mean queue depth calculation (0 to 15).

Step 8	**end** **Example:** `Device(config-pmap-c)# end`	Exits class map configuration mode and returns to privileged EXEC mode.

Configure class-based packet marking for ingress policy-map

Before you begin

You should have created a class map and a policy map before beginning this procedure.

This procedure explains how to configure these class-based packet marking features on your device:

- CoS value
- Discard-class value
- DSCP value
- IP value
- MPLS experimental value
- Precedence value
- QoS group value
- Traffic-class value

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map** *policy name*
	Example:
	`Device(config)# policy-map policy1`
	`Device(config-pmap)#`
		Enters policy map configuration mode.
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy.

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class class-default`
	`Device(config-pmap-c)#`
		Enters policy class map configuration mode. Specifies the name of the class whose policy you want to create or change.
		Command options for policy class map configuration mode include these:
		• **exit**: Exits from the QoS class action configuration mode.
		• **no**: Negates orsets default values for the command.
		• **police**: Policer configuration options.
		• **service-policy**: Configures the QoS service policy.
		• **set**: Sets QoS values using these options:
		  • CoS values

	Command or Action	Purpose
		• Discard-class values • DSCP values • IP values • MPLS values • Precedence values • QoS group values • Traffic class values **Note** This procedure describes the supported configurations using **set** command options. The other command options (**bandwidth**) are described in other sections of this guide. Only one **set**command for QoS tag (CoS/DSCP/PREC/EXP) is supported per class. The same class can also have set traffic-class and/or set discard-class if it is a ingress policy.

Step 4	**set cos** {*cos value* | **cos table** *table-map name* | **dscp table** *table-map name* | **precedence** *table-map name* | **qos-group table** *table-map name*}
	
	Example:
	
	`Device(config-pmap)# set cos cos table cos-mapping`
	`Device(config-pmap)#`
		(Optional) Sets the specific IEEE 802.1Q Layer 2 CoS value of an outgoing packet. Values are from 0 to 7.
		
		You can also set the CoS values using a table map. This is a list of from types supported when using atable map. For example, **set cos qos-group table** *table-map name* means use the table-map to map from QoS-group to CoS.
		
		• **cos table**: Sets the CoS value based on a table map.
		• **dscp table**: Sets the CoS value from DSCP value based on a table map.
		• **precedence table**: Sets the CoS value from precedence value based on a table map.
		• **qos-group table**: Sets the CoS value from QoS group based on a table map.
		
		Note
		• **set cos qos-group table** is only supported in egress policy.
		• Table-map is only supported in class-default.

	Command or Action	Purpose

Step 5	**set discard-class** {*discard-class value* | **cos table** *table-map name* | **dscp table** *table-map name* | **mpls experimental imposition table** *table-map name* | **precedence** *table-map name*}
	Example:
	`Device(config-pmap)# set discard-class 0`
	`Device(config-pmap)#`
		(Optional) Sets discard-class values from 0 to 1. You can also set the discard-class values using a table map.
		• **cos table**: Sets the discard-class value from CoS value based on a table map.
		• **dscp table**: Sets the discard-class value from DSCP value based on a table map.
		• **mpls experimental imposition table**: Sets the discard-class value from MPLS experimental based on a table map.
		• **precedence table**: Sets the discard-class value from precedence value based on a table map.
		Note
		• Table-map is only supported in class-default.
		• If the discard-class value is set to 1 and a policer is configured in the same class, the discard-class will not have any impact during congestion scenario, as policer (color blind) marks all confirmed packets as green.

Step 6	**set dscp** {*dscp value* | **cos table** *table-map name* | **default** | **dscp table** *table-map name* | **ef** | **precedence table** *table-map name* | **qos-group table** *table-map name*}
	Example:
	`Device(config-pmap)# set dscp af11`
	`Device(config-pmap)#`
		(Optional) Sets the DSCP value. Values are from 0 to 63.
		You can also set the DSCP values using a table map. This is a list of from types supported when using a table map.
		For example, **set dscp qos-group table** *table-map name* means use the table- map to map from QoS-group to DSCP.
		• **cos table**: Sets the packet DSCP value from CoS value based on a table map.
		• **dscp table**: Sets the packet DSCP value from DSCP based on a table map.
		• **precedence table**: Sets the packet DSCP value from precedence based on a table map.
		• **qos-group table**: Sets the packet DSCP value from a QoS group based upon a table map.

	Command or Action	Purpose
		Note Table-map is only supported in class-default.

Step 7	**set ip** {**dscp** | **precedence**}
	Example:
	`Device(config-pmap)# set ip dscp c3`
	`Device(config-pmap)#`
		(Optional) Sets IP specific values. These values are either IP DSCP or IP precedence values.
		You can also set the IP values using a table map. This is a list of from types supported when using a table map. For example, **set qos-group dscp table** *table-map name* means use the table-map to map from DSCP value to QoS-group.
		Note
		Table-map is only supported in class-default.
		You can set these values using the **set ip dscp** command:
		• **cos table**: Sets the packet DSCP value from CoS value based on a table map.
		• **dscp table**: Sets the packet DSCP value from DSCP based on a table map.
		• **precedence table**: Sets the packet DSCP value from precedence based on a table map.
		• **qos-group table**: Sets the packet DSCP value from a QoS group based upon a table map.
		This is a list of from types supported when using a table map. For example, **set ip precedence qos-group table** *table-map name* means use the table-map to map from QoS-group to IP precedence.
		• **cos table**: Sets the packet precedence value from CoS value based on a table map.
		• **dscp table**: Sets the packet precedence from DSCP value based on a table map.
		• **precedence table**: Sets the precedence value from precedence based on a table map.
		• **qos-group table**: Sets the precedence value from a QoS group based upon a table map.

	Command or Action	Purpose
		Note **set ip precedence qos-group table** *table-mapname* is only supported in egress.

Step 8	**set mpls experimental** {**dscp table** *table-map name* | **imposition** {*experimental value* | **dscp table** *table-map name* | **precedence table** *table-map name*} | **precedence table** *table-map name* | **topmost** {*experimental value* | **cos** *table-map name* | **precedence** *table-map name* | **qos-group** *table-map name*}}
	Example:
	`Device(config-pmap)# set mpls experimental imposition 4`
	`Device(config-pmap)#`
		(Optional) Sets the MPLS experimental values from 0 to 7. You can also set the MPLS values using a table map.
		• **dscp table**: Sets the MPLS experimental value from DSCP value on a table map.
		• **precedence table**: Sets the MPLS experimental value from precedence based on a table map.
		• **cos table**: Sets the MPLS experimental value from CoS value based on a table map.
		• **qos-group table**: Sets the MPLS experimental from a QoS group based upon a table map.

Step 9	**set precedence** {*precedence value* | **cos table** *table-map name* | **dscp table** *table-map name* | **precedence table** *table-map name* | **qos-group table** *table-map name* | **traffic-class table** *table-map name*}
	Example:
	`Device(config-pmap)# set precedence 5`
	`Device(config-pmap)#`
		(Optional) Sets precedence values in IPv4 and IPv6 packets. Values are from 0 to 7.
		You can also set the precedence values using a table map. This is a list of from types supported when using a table map. For example, **set precedence qos-group table** *table-mapname* means use the table-map to map from QoS-group to precedence.
		• **cos table**: Sets the packet precedence value from CoS value based on a table map.
		• **dscp table**: Sets the packet precedence value from DSCP value on a table map.
		• **precedence table**: Sets the precedence value from precedence based on a table map.
		• **qos-group table**: Sets the precedence value from a QoS group based upon a table map.
		Note Table-map is only supported in class-default.

	Command or Action	Purpose

Step 10	**set qos-group** {*qos-group value* | **cos table** *table-map name* | **dscp table** *table-map name* | **ip-precedence** | **mpls experimental table** *table-map name* | **precedence table** *table-map name*}
	Example:
	`Device(config-pmap)# set qos-group 10`
	`Device(config-pmap)#`
		(Optional) Sets QoS group values from 0 to 30. You can also set the QoS group using a table map.
		• **cos table**: Sets QoS group from CoS value based on a table map.
		• **dscp table**: Sets QoS group from DSCP value based on a table map.
		• **mpls experimental table**: Sets QoS group from MPLS experimental value based on a table map.
		• **precedence table**: Sets QoS group from precedence based on a table map.

Step 11	**set traffic-class** {*traffic-class value* | **cos table** *table-map name* | **dscp table** *table-map name* | **mpls experimental imposition table** *table-map name* | **precedence table** *table-map name*}
	Example:
	`Device(config-pmap)# set traffic-class 3`
	`Device(config-pmap)#`
		(Optional) Sets traffic-class values from 0 to 7, or from these table-maps:
		• **cos table**: Sets the traffic-class value from COS value based on a table map.
		• **dscp table**: Sets the traffic-class valuefrom DSCP value based on a tablemap.
		• **mpls experimental imposition table**: Sets the traffic-class value from MPLS experimental value based on a table map.
		• **precedence table**: Sets the traffic-class value from precedence based on a table map.

Step 12	**end**
	Example:
	`Device(config-pmap)# end`
	`Device#`
		Exits policy map configuration mode and returns to privileged EXEC mode.

Step 13	**show policy-map** **Example:** `Device# show policy-map`	(Optional) Displays policy configuration information for all classes configured for all service policies.

What to do next

Attach the traffic policy to an interface using the **service-policy** command.

Attach a traffic policy to an interface

Before you begin

A traffic class and traffic policy must be created before attaching a traffic policy to an interface.

After the traffic class and traffic policy are created, you must use the **service-policy** interface configuration command to attach a traffic policy to an interface, and to specify the direction in which the policy should be applied (either on packets coming into the interface or packets leaving the interface).

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**interface** *type*
	Example:
	`Device(config)# interface HundredGigE4/0/1`
	`Device(config-if)#`
		Enters interface configuration mode and configures an interface.
		Command parameters for the interface configuration include:
		• **TenGigabitEthernet**: 10-Gigabit interface numbering
		• **FiftyGigabitEthernet**: 1/10/25/50-Gigabit interface numbering
		• **HundredGigabitEthernet**: 140/100-Gigabit interface numbering
		• **FourHundredGigabitEthernet**: 40/100/400-Gigabit interface numbering
		Note
		• Tunnel and VLAN interface is not supported.
		• Policing, queuing, and marking are not supported on the management interface.

Step 3	**service-policy** [**type queueing**] {**input** *policy-map* | **output** *policy-map*} **Example:** **Note** The output direction is only for remarks on non-queueing policies.	Attaches a policy map to an input or output interface. This policy map is then used as the service policy for that interface. In this example, the traffic policy evaluates all traffic leaving that interface. **Note**

	Command or Action	Purpose
	Device(config-if)# **service-policy input policy_map_01** Device(config-if)#	• Non-queuing policy with policer is supported only in ingress. • On egress, only non-queueing policy with marking is supported. • Queuing policy (**type queueing**) is supported only in egress. • Meter police should be applied in ingress direction and remark police should be applied in egress direction.
Step 4	**end** **Example:** Device(config-if)# **end** Device#	Exits interface configuration mode and returns to privileged EXEC mode.
Step 5	**show policy-map** **Example:** Device# **show policy-map**	(Optional) Displays statistics for the policy on the specified interface.

What to do next

Proceed to attach any other traffic policy to an interface, and to specify the direction in which the policy should be applied.

Classify, police, and mark traffic on physical ports by using policy maps

Before you begin

You should have already decided upon the classification, policing, and marking of your network traffic by policy maps prior to beginning this procedure.

You can configure a nonhierarchical policy map on a physical port that specifies which traffic class to act on. Actions supported are remarking and policing.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:**	Enters global configuration mode.

	Command or Action	Purpose
	`Device# configure terminal`	

Step 2	**class-map** {*class-map name* | **match-any** | **match-all**}
	Example:
	`Device(config)# class-map ipclass1`
	`Device(config-cmap)# exit`
	`Device(config)#`
		Enters class-map configuration mode.
		• Creates a class-map to be used for matching packets to the class whose name you specify.
		• If you specify **match-any**, one of the match criteria must be met for traffic entering the traffic class to be classified as part of the traffic class. This is the default.
		• If you specify **match-all**, all of the match criteria must be met for traffic entering the traffic class to be classified as part of the traffic class.
		Note
		This is the default. If **match-any** or **match-all** is not explicitly defined, **match-all** is chosen by default.

Step 3	**match access-group** *access list name*
	Example:
	`Device(config-cmap)# match access-group acl`
	`Device(config-cmap)# exit`
	`Device(config)#`
		The following parameters are available for this command:
		• access-group
		• cos
		• dscp
		• ip
		• precedence
		• qos-group

Step 4	**policy-map** *policy-map-name*
	Example:
	`Device(config)# policy-map ipclass1`
	`Device(config-pmap)#`
		Creates a policy map by entering the policy map name, and enters policy-map configuration mode.
		By default, no policy maps are defined.

Step 5	**class** {*class-map-name* | **class-default**}
	Example:
	`Device(config-pmap)# class ipclass1`
	`Device(config-pmap-c)#`
		Defines a traffic classification, and enter policy-map class configuration mode.
		By default, no policy map class-maps are defined.

	Command or Action	Purpose
		If a traffic class has already been defined by using the **class-map** global configuration command, specify its name for *class-map-name* in this command. A **class-default** traffic class is predefined and can be added to any policy. It is always placed at the end of a policy map. With an implied **match any** included in the **class-default** class, all packets that have not already matched the other traffic classes will match **class-default**.

Step 6	**set** {**cos** | **discard-class** | **dscp** | **ip** | **mpls** | **precedence** | **qos-group** | **traffic-class**}
	
	Example:
	
	`Device(config-pmap-c)# set dscp 45`
	`Device(config-pmap-c)#`
		(Optional) Sets the QoS values. Possible QoS configuration values include:
		
		• **cos**: Sets the IEEE 802.1Q/ISL class of service/user priority.
		• **discard-class**: Sets discard behavior identifier.
		• **dscp**: Sets DSCP in IP(v4)and IPv6 packets.
		• **ip**: Sets IP specific values.
		• **mpls**: Sets MPLS specific values.
		• **precedence**: Sets precedence in IP(v4) and IPv6 packet.
		• **qos-group**: Sets the QoS group.
		• **traffic-class**: Sets the traffic class.
		
		In this example, the **set dscp** command classifies the IP traffic by setting a new DSCP value in the packet.

Step 7	**police** {*target_bit_rate* | **cir** | **rate**}
	
	Example:
	
	`Device(config-pmap-c)# police 1200000 conform-action transmit exceed-action drop`
	`Device(config-pmap-c)#`
		(Optional) Configures the policer:
		
		• *target_bit_rate*: Specifies the bit rate per second, enter a value between 1200000 and 400000000000.
		• **cir**: Committed Information Rate.
		• **rate**: Specifies the police rate PCR for hierarchical policies.
		
		In this example, the **police** command adds a policer to the class where any traffic beyond the 1200000 set target bit rate is dropped.

	Command or Action	Purpose
Step 8	**exit** **Example:** `Device(config-pmap-c)# exit`	Returns to policy map configuration mode.
Step 9	**exit** **Example:** `Device(config-pmap)# exit`	Returns to global configuration mode.
Step 10	**interface** *interface-id* **Example:** `Device(config)# interface HundredGigE 1/0/2`	Specifies the port to attach to the policy map, and enters interface configuration mode. Valid interfaces include physical ports.
Step 11	**service-policy input** *policy-map-name* **Example:** `Device(config-if)# service-policy input ipclass1`	Specifies the policy-map name, and applies it to an ingress port. Only one policy map per ingress port is supported.
Step 12	**end** **Example:** `Device(config-if)# end`	Returns to privileged EXEC mode.
Step 13	**show policy-map** [*policy-map-name* [**class** *class-map-name*]] **Example:** `Device# show policy-map ipclass1`	(Optional) Verifies your entries.
Step 14	**copy running-config startup-config** **Example:** `Device# copy-running-config startup-config`	(Optional) Saves your entries in the configuration file.

Classify and mark traffic by using policy maps

Before you begin

You should have already decided upon the classification, policing, and marking of your network traffic by using policy maps prior to beginning this procedure.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**class-map** {*class-map name* | **match-any** | **match-all**}
	Example:
	`Device(config)# class-map class_cos2`
	`Device(config-cmap)# match cos 2`
		Enters class map configuration mode.
		• Creates a class map to be used for matching packets to the class whose name you specify.
		• If you specify **match-any**, one of the match criteria must be met for traffic entering the traffic class to be classified as part of the traffic class.
		• If you specify **match-all**, all of the match criteria must be met for traffic entering the traffic class to be classified as part of the traffic class.
		Note
		This is the default. If **match-any** or **match-all** is not explicitly defined, **match-all** is chosen by default.

Step 3	**policy-map** *policy-map-name*
	Example:
	`Device(config)# policy-map policy_cos2`
	`Device(config-pmap)#`
		Creates a policy map by entering the policy map name, and enters policy-map configuration mode.
		By default, no policy maps are defined.

Step 4	**description** *description* **Example:** `Device(config-pmap)# description cos 2`	(Optional) Enters a description of the policy map.

	Command or Action	Purpose

Step 5	**class** {*class-map-name* | **class-default**}
	Example:
	`Device(config-pmap)# class class_cos2`
	`Device(config-pmap-c)#`
		Defines a traffic classification, and enters the policy-map class configuration mode.
		By default, no policy map class-maps are defined.
		If a traffic class has already been defined by using the **class-map** global configuration command, specify its name for *class-map-name* in this command.
		A **class-default** traffic class is predefined and can be added to any policy. It is always placed at the end of a policy map. With an implied **match any** included in the **class-default** class, all packets that have not already matched the other traffic classes will match **class-default**.

Step 6	**set** {**cos** | **qos-group** | **traffic-class** | **discard-class**}
	Example:
	`Device(config-pmap-c)# set cos 7`
	`Device(config-pmap-c)#`
		(Optional) Sets the QoS values. Possible QoS configuration values include:
		• **cos**: Sets the IEEE 802.1Q/ISL class of service/user priority.
		• **qos-group**: Sets QoS group.
		• **traffic-class**: Sets traffic class.
		• **discard-class**: Sets discard class.
		In this example, the **set cos** command classifies the traffic by matching the packets with a COS values of 2, and sets the CoS value to a new value of 7.

Step 7	**exit** **Example:** `Device(config-pmap-c)# exit`	Returns to policy map configuration mode.
Step 8	**exit** **Example:** `Device(config-pmap)# exit`	Returns to global configuration mode.
Step 9	**interface** *interface-id* **Example:** `Device(config)# interface HundredGigE1/0/3`	Specifies the port to attach to the policy map, and enters interface configuration mode. Valid interfaces include physical ports, which should be a trunk port for Layer 2 based classification.

	Command or Action	Purpose
Step 10	**service-policy input** *policy-map-name* **Example:** `Device(config-if)# service-policy input policy_vlan100`	Specifies the policy-map name, and applies it to an ingress port. Only one policy map per ingress port is supported.
Step 11	**end** **Example:** `Device(config-if)# end`	Returns to privileged EXEC mode.
Step 12	**show policy-map** [*policy-map-name* [**class** *class-map-name*]] **Example:** `Device# show policy-map`	(Optional) Verifies your entries.
Step 13	**copy running-config startup-config** **Example:** `Device# copy-running-config startup-config`	(Optional) Saves your entries in the configuration file.

Configure table maps

Table maps area form of marking, and also enable the mapping and conversion of one field to another using a table. For example, a table map can be used to map and convert a Layer 2 CoS setting to a new or same CoS value, and also convert a Layer 2 CoS setting to a traffic class or a QoS group. Similar settings apply to Layer 3 DSCP and Precedence.

When setting a table-map in a policy, the **from** and **to** values should be of the same type, example DCSP to DSCP, COS to COS, and so on. This exception is not applicable for QoS- group and traffic-class where **from** and **to** values are of different types, for example, DSCP to traffic-class, COS to traffic-class, and so on.

Note

- A table map can be referenced in multiple policies or multiple times in the same policy.
- The examples in this procedure uses table maps to map DSCP value to traffic-class.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**table-map** *name* {**default** {*default value* | **copy** | **ignore**} | **exit** | **map** {**from** *from value* **to** *to value* } | **no**}
	Example:
	`Device(config)# table-map table01`
	`Device(config-tablemap)#`
		Creates a table map and enters the table map configuration mode. In table map configuration mode, you can perform the following tasks:
		• **default**: Configures the table map default value, or sets the default behavior for a value not found in the table map to copy or ignore.
		• **exit**: Exits from the table map configuration mode.
		• **map**: Maps a *from* to a *to* value in the table map.
		• **no**: Negates or sets the default values of the command.

Step 3	**map from** *value* **to** *value*
	Example:
	`Device(config-tablemap)# map from 0 to 2`
	`Device(config-tablemap)# map from 1 to 4`
	`Device(config-tablemap)# map from 24 to 3`
	`Device(config-tablemap)# map from 40 to 6`
	`Device(config-tablemap)# default 0`
	`Device(config-tablemap)#`
		In this step, packets with DSCP value 0 are marked to the traffic-class 2, DSCP value 1 to the traffic-class 4, DSCP value 24 to the traffic-class 3, DSCP value 40 to the traffic-class 6 and all others to the traffic-class 0.
		Note
		The mapping from DSCP to traffic-class values in this example is configured by using the **set** policy map class configuration command as described in a later step in this procedure.

Step 4	**exit**
	Example:
	`Device(config-tablemap)# exit`
	`Device(config)#`
		Returns to global configuration mode.

Step 5	**exit** **Example:**	Returns to privileged EXEC mode.

	Command or Action	Purpose
	`Device(config) exit` `Device#`	

Step 6	**show table-map**
	Example:
	`Device# show table-map`
	`Table Map table01`
	`from 0 to 2`
	`from 1 to 4`
	`from 24 to 3`
	`from 40 to 6`
	`default 0`
		Displays the table map configuration.

Step 7	**configure terminal**
	Example:
	`Device# configure terminal`
	`Device(config)#`
		Enters global configuration mode.

Step 8	**policy-map**
	Example:
	`Device(config)# policy-map table-policy`
	`Device(config-pmap)#`
		Configures the policy map for the table map.

Step 9	**class class-default**
	Example:
	`Device(config-pmap)# class class-default`
	`Device(config-pmap-c)#`
		Matches the class to the system default.

Step 10	**set traffic-class dscp table** *table map name*
	Example:
	`Device(config-pmap-c)# set traffic-class dscp table table01`
	`Device(config-pmap-c)#`
		Incoming packets with DSCP value are mapped to hit a particular VOQ which is defined by the traffic-class values in a table-map.

Step 11	**end**
	Example:
	`Device(config-pmap-c)# end`
	`Device#`
		Returns to privileged EXEC mode.

What to do next

Configure any additional policy maps for QoS for your network. After creating your policy maps, attach the traffic policy or polices to an interface using the **service-policy** command.

Configure QoS functions

The following sections provide configurational information about QoS features and functionality.

Configure bandwidth

Before you begin

You should have created a created a class-map for bandwidth based on traffic-class classification before beginning this procedure.

This procedure explains how to configure bandwidth on your device.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map type queueing** *policy name*
	Example:
	`Device(config)# policy-map type queueing policy_bandwidth01`
	`Device(config-pmap)#`
		Enters policy map configuration mode.
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy.

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class class_bandwidth01`
	`Device(config-pmap-c)#`
		Enters policy class map configuration mode. Specifies the name of the class whose policy you want to create or change. Command options for policy class map configuration mode include the following:
		• *word*: Class map name.
		• **class-default**: In type queueing policy, class-default always matches traffic-class 0.

Step 4	**bandwidth remaining ratio** *ratio* **Example:**	Configures the bandwidth for the policy map. • *ratio*: Ratios can range from 1 to 63.

	Command or Action	Purpose
	`Device(config-pmap-c)# bandwidth remaining ratio 10` `Device(config-pmap-c)#`	**Note** Bandwidth command is applicable from traffic class 6 to traffic class 0 classification and should not be defined with priorities.

Step 5	**end**
	Example:
	`Device(config-pmap-c)# end`
	`Device#`
		Saves configuration changes.

Step 6	**show policy-map** **Example:** `Device# show policy-map`	(Optional) Displays policy configuration information for all classes configured for all service policies.

What to do next

Configure any additional policy maps for QoS for your network. After creating the policy maps, attach the traffic policy or polices to an interface using the **service-policy** command.

Configure police

Before you begin

You should have created a class map for policing before beginning this procedure.

This procedure explains how to configure policing on your device.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map** *policy name*
	Example:
	`Device(config)# policy-map policy_police01`
	`Device(config-pmap)#`
		Enters policy map configuration mode.
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy.

	Command or Action	Purpose

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class class_police01`
	`Device(config-pmap-c)#`
		Enters policy class map configuration mode. Specifies the name of the class whose policy you want to create or change. Command options for policy class map configuration mode include the following:
		• *word*: Class map name.
		• **class-default**: System default class matching any otherwise unclassified packets.

Step 4	**police** {*target_bit_rate* [**conform-action** | **pir**] | **cir** {*target_bit_rate* | **percent** *percentage*} | **rate** {*target_bit_rate* | **percent** *percentage*} **conform-action transmit exceed-action** {**drop** [**violate action**] | **set-cos-transmit** | **set-dscp-transmit** | **set-prec-transmit** | **transmit** [**violate action**]}}
	Example:
	`Device(config-pmap-c)# police 1200000 conform-action transmit exceed-action drop`
	`Device(config-pmap-c)#`
		The following **police** subcommand options are available:
		• *target_bit_rate*: Bits per second (from 1200000 to 400000000000).
		• **conform-action**: Action taken when rate is less than conform burst.
		• **pir**: Peak Information Rate.
		• **cir**: Committed Information Rate.
		• *target_bit_rate*: Target bit rate (from 1200000 to 400000000000).
		• **percent**: Percentage of interface bandwidth for CIR.
		• **rate**: Specifies the police rate, PCR for hierarchical policies, or SCR for single-level ATM 4.0 policer policies.
		• *target_bit_rate*: Target bit rate (from 1200000 to 400000000000).
		• **percent**: Percentage of interface bandwidth for rate.
		The following **police conform-action transmit exceed-action** subcommand options are available:
		• **drop**: Drops the packet.
		• **set-cos-transmit**: Sets the CoS value and sends it.
		• **set-discard-class- transmit**: Sets the discard-class value and sends it.

	Command or Action	Purpose
		• **set-dscp-transmit**: Sets the DSCP value and sends it. • **set-prec-transmit**: Rewrites the packet precedence and sends it. • **transmit**: Transmits the packet. **Note** Policer-based markdown actions are only supported using table maps. Only one markdown table map is allowed for each marking field in the device.

Step 5	**end**
	Example:
	`Device(config-pmap-c)# end`
	`Device#`
		Saves configuration changes.

Step 6	**show policy-map** **Example:** `Device# show policy-map`	(Optional) Displays policy configuration information for all classes configured for all service policies. **Note** The **show policy-map** command output does not display counters for conformed bytes and exceeded bytes

What to do next

Configure any additional policy maps for QoS for your network. After creating your policy maps, attach the traffic policy or polices to an interface using the **service-policy** command.

Configure priority

Before you begin

You should have created a class map for priority before beginning this procedure.

This procedure explains how to configure priority on your device.

Note The device supports giving priority to specified queues.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map type queueing** *policy name*
	Example:
	`Device(config)# policy-map type queueing policy_priority01`
	`Device(config-pmap)#`
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy, and enters policy map configuration mode.

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class traffic-class7`
	`Device(config-pmap-c)#`
		Specifies the name of the class whose policy you want to create or change, and enters policy class map configuration mode. Command options for policy class map configuration mode include these:
		• *word*: Class map name.
		• **class-default**: In type queueing policy, class-default always matches traffic-class 0.

Step 4	**priority level** *level_value*
	Example:
	`Device(config-pmap-c)# priority level 1`
	`Device(config-pmap-c)#`
		(Optional) This command assigns a strict scheduling priority for the class.
		• *level_value*: Specifies the priority queue.
		Note
		Shaping or policing a priority queue limits the traffic to the configured rate regardless of queue utilization of non-priority queues.

Step 5	**end**
	Example:
	`Device(config-pmap-c)# end`
	`Device#`
		Saves configuration changes.

Step 6	**show policy-map type queueing** **Example:** `Device# show policy-map type queueing`	(Optional) Displays the runtime representation and statistics of all the queueing policies configured on the device.

What to do next

Configure any additional policy maps for QoS for your network. After creating your policy maps, attach the traffic policy or polices to an interface using the **service-policy** command.

Configure queues and shaping

These sections provide configurational information about queueing and shaping.

Configure egress queue characteristics

Depending on the complexity of your network and your QoS solution, you may need to perform all of the procedures in this section. You need to make decisions about these characteristics:

- Which packets are mapped by DSCP, CoS, or QoS group value to each queue and threshold ID?
- Which traffic class based classification apply to the queues, and if incoming packets are mapped to any of the traffic classes from 0 to 7, so that traffic gets queued in specific VoQ queues?
- How much of the fixed buffer space is allocated to the queues?
- Does the bandwidth of the port need to be rate limited?
- How often should the egress queues be serviced and which technique (shaped, shared, or both) should be used?



Note You can only configure the egress queues on the device, and only traffic-class based classification is supported.

Configure queue limits

Queue-limit can be only be configured in unit of bytes, and total number of thresholds are limited.

Before you begin

These are prerequisites for this procedure:

- You should have created a class map for the queue limits before beginning this procedure.
- You must have configured either bandwidth, shape, or priority on the policy map prior to configuring the queue limits.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:**	Enters global configuration mode.

	Command or Action	Purpose
	`Device# configure terminal`	

Step 2	**policy-map type queueing** *policy name*
	Example:
	`Device(config)# policy-map type queueing policy_queuelimit01`
	`Device(config-pmap)#`
		Specifies the name of the queueing profile policy and enters policy map configuration mode.

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class traffic-class7`
	`Device(config-pmap-c)#`
		Specifies the name of the class to be associated with the policy and enters policy class map configuration mode. Command options for policy class map configuration mode include these:
		• *word*: Class map name.
		• **class-default**: In type queueing policy, class-default always matches traffic-class 0.

Step 4	**shape average** {*bits/s* | **percent**}
	Example:
	`Device(config-pmap-c)# shape average 1000000000`
	`Device(config-pmap-c)#`
		Configures the traffic shaping average. The parameters include:
		• *bits/s*: Use this command to configure a specific value. The range is 1200000 to 400000000000.
		• **percent**: Allocates a maximum bandwidth to a particular class. The value can be from 1 to 100. Shaping is done on the percentage value, and traffic will not exceed the shape percentage value.

Step 5	**queue-limit** *value* [**bytes**] **Example:** `Device(config-pmap-c)# queue-limit 1000000 bytes`	Sets the queue limit threshold percentage value in bytes.

Step 6	**end**
	Example:
	`Device(config-pmap-c)# end`
	`Device#`
		Exits policy class map configuration mode and enters privileged EXEC mode.

Configure shaping

Before you begin

You should have created a class map for shaping before beginning this procedure.

You use the **shape** command to configure shaping (maximum bandwidth) for a particular class. The queue's bandwidth is restricted to this value even though the port has additional bandwidth left. You can configure shaping as an average percent, as well as a shape average value in bits per second.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map type queuing** *policy name*
	Example:
	`Device(config)# policy-map type queuing policy_shaping01`
	`Device(config-pmap)#`
		Enters policy map configuration mode.
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy.

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class traffic-class7`
	`Device(config-pmap-c)#`
		Enters policy class map configuration mode. Specifies the name of the class whose policy you want to create or change. Command options for policy class map configuration mode include these:
		• *word*: Class map name.
		• **class-default**: In type queueing policy,class-default always matches traffic-class 0.

Step 4	**shape average** {*target bit rate* | **percent** *percentage*}
	Example:
	`Device(config-pmap-c)# shape average percent 50`
	`Device(config-pmap-c)#`
		Configures the average shape rate. You can configure the average shape rate by target bit rates (bits per second) or by percentage of interface bandwidth for the Committed Information Rate (CIR).

Step 5	**end** **Example:** `Device(config-pmap-c)# end`	Saves configuration changes.

	Command or Action	Purpose
	`Device#`	
Step 6	**show policy-map** **Example:** `Device# show policy-map`	(Optional) Displays policy configuration information for all classes configured for all service policies.

What to do next

Configure any additional policy maps for QoS for your network. After creating your policy maps, attach the traffic policy or polices to an interface using the **service-policy** command.

Configure shaper profile queuing

This procedure explains how to configure shaper profile queuing on your switch:

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.

Step 2	**policy-map type queueing** *policy name*
	Example:
	`Device(config)# policy-map type queueing policy_shaping01`
	`Device(config-pmap)#`
		Enters policy map configuration mode.
		Creates or modifies a policy map that can be attached to one or more interfaces to specify a service policy.
		policy-map-name is the name of the child policy map. The name can be a maximum of 40 alphanumeric characters.

Step 3	**class** *class name*
	Example:
	`Device(config-pmap)# class traffic-class1`
	`Device(config-pmap-c)#`
		Enters policy class map configuration mode. Specifies the name of the class whose policy you want to create or change. Command options for policy class map configuration mode include these:
		• *word*: Class map name.
		• **class-default**: System default class matching any otherwise unclassified packets.

	Command or Action	Purpose

Step 4	**bandwidth remaining ratio** *ratio*
	Example:
	`Device(config-pmap-c)# bandwidth remaining ratio 10`
	`Device(config-pmap-c)#`
		(Optional) Configures ratio for sharing excess bandwidth.The range is from 1 to 63.

Step 5	**shape average** {*target bit rate* | **percent** *percentage*}
	Example:
	`Device(config-pmap-c)# shape average percent 50`
	`Device(config-pmap-c)#`
		Configures the average shape rate. You can configure the average shape rate by target bit rates (bits per second) or by percentage of interface bandwidth for the Committed Information Rate (CIR).

Step 6	**end**
	Example:
	`Device(config-pmap-c)# end`
	`Device#`
		Saves configuration changes.

Shaper profile queuing configuration

The following is the example for shaper queuing:

```
Device# show policy-map type queueing parent class class-default

Policy Map type queueing class-default
    Class class-default
      bandwidth remaining ratio 10
```

Monitor QoS

These commands can be used to monitor QoS on the device:

Table 9: Monitoring QoS

Command	Description
show class-map [*class_map_name*]	Displays a list of all class maps configured.
show policy-map [**type queueing**] [*policy_map_name*]	Displays a list of all policy maps configured. Command parameters include: • policy map name • type queueing

Command	Description
show policy-map interface {**TenGigE** | **HundredGigE**}	Displays the runtime representation and statistics of all the policies configured on the device. Command parameters include: • TenGigabitEthernet: 10-Gigabit Ethernet **Note** This is not supported in Cisco C9610 Series Smart Switches and Cisco C9610 Series Smart Switches. • FiftyGigabitEthernet: 1-Gigabit Ethernet, 25-Gigabit Ethernet, 50-Gigabit Ethernet • HundredGigabitEthernet: 100-Gigabit Ethernet, 40-Gigabit Ethernet • FourHundredGigabitEthernet: 400-Gigabit Ethernet **Note** Though **wireless** option is visible on the CLI, it is not supported.
show policy-map type queueing interface {**TenGigE** | **HundredGigE**}	Displays the runtime representation and statistics of all the queuing policies configured on the device. Command parameters include: • TenGigabitEthernet: 10-Gigabit Ethernet • FiftyGigabitEthernet: 50-Gigabit Ethernet, 25-Gigabit Ethernet • HundredGigabitEthernet: 100-Gigabit Ethernet, 40-Gigabit Ethernet • FourHundredGigabitEthernet: 400-Gigabit Ethernet **Note** Though **wireless** option is visible on the CLI, it is not supported.
show table-map	Displays all the table maps and their configurations.

Examples: TCP protocol classification

TCP packets can be classified based on port numbers. The configuration for TCP protocol is as follows:

```
Device# show ip access-list tcp

Extended IP access list tcp
```

```
10 permit tcp any any eq 80

Device# show run class-map tcp

Current configuration : 63 bytes
!
class-map match-all tcp
match access-group name tcp
!
end

Device# show run policy-map tcp

Current configuration : 56 bytes
!
policy-map tcp
class tcp
police 1000000000
!
end

Device# show run interface fif 1/0/1

Current configuration : 93 bytes
!
interface FiftyGigE1/0/1
no ip address
no keepalive
service-policy input tcp
end
```

Examples: UDP protocol classification

UDP packets can be classified based on port numbers. The configuration example for UDP protocol is as follows:

```
Device# show ip access-list udp

Extended IP access list udp
10 permit udp any any eq ntp

Device# show run class-map udp

Building configuration...

Current configuration : 63 bytes
!
class-map match-all udp
match access-group name udp
!
end

Device# show run policy-map udp

Building configuration...

Current configuration : 56 bytes
!
policy-map udp
class udp
police 1000000000
!
```

```
end

Device# show run interface fif 1/0/1

Current configuration : 93 bytes
!
interface FiftyGigE1/0/1
no ip address
no keepalive
service-policy input udp
end
```

Examples: RTP protocol classification

RTP packets can be classified based on port numbers. The configuration example for RTP protocol is as follows:

```
Device# show ip access-list rtp

Extended IP access list rtp
10 permit udp any any eq 554
11 permit tcp any any eq 554

Device# show run class-map rtp

Current configuration : 63 bytes
!
class-map match-all rtp
match access-group name rtp
!
end

Device# show run policy-map rtp

Current configuration : 56 bytes
!
policy-map rtp
class rtp
police 1000000000
!
end

Device# show run int FiftyGigE1/0/1

Current configuration : 93 bytes
!
interface FiftyGigE1/0/1
no ip address
no keepalive
service-policy input rtp
end
```

Examples: Classification by ACLs

This example shows how to classify packets for QoS by using ACLs:

```
Device# configure terminal
Device(config)# ip access-list extended aclv4
Device(config-ext-nacl)# permit udp host 150.1.1.1 host 14.1.1.1
Device(config)# class-map aclv4
Device(config-cmap)# match access-group name aclv4
```

After creating a class map by using an ACL, you then create a policy map for the class, and apply the policy map to an interface for QoS.

Examples: Class of service Layer 2 classification

This example shows how to classify packets for QoS using a class of service Layer 2 classification:

```
Device# configure terminal
Device(config)# class-map match-any cos
Device(config-cmap)# match cos ?

<0-7> Enter up to 4 class-of-service values separated by white-spaces

Device(config-cmap)# match cos 3 4 5
Device(config-cmap)#
```

After creating a class map by using a CoS Layer 2 classification, you then create a policy map for the class, and apply the policy map to an interface for QoS.

Examples: Class of service DSCP classification

This example shows how to classify packets for QoS using a class of service DSCP classification:

```
Device# configure terminal
Device(config)# class-map match-any dscp
Device(config-cmap)# match dscp af21 af22 af23
Device(config-cmap)#
```

After creating a class map by using a DSCP classification, you then create a policy map for the class, and apply the policy map to an interface for QoS.

Examples: Classification by DSCP or precedence values

This example shows how to classify packets by using DSCP or precedence values:

```
Device# configure terminal
Device(config)# class-map prec2
Device(config-cmap)# description matching precedence 2 packets
Device(config-cmap)# match ip precedence 2
Device(config-cmap)# exit
Device(config)# class-map ef
Device(config-cmap)# description EF traffic
Device(config-cmap)# match ip dscp ef
Device(config-cmap)# exit

Device(config)# class-map prec5
Device(config-cmap)# match precedence 5
Device(config-cmap)# exit

Device(config)# class-map cs1
Device(config-cmap)# match dscp cs1
Device(config-cmap)#
```

After creating a class map by using a DSCP or precedence values, you then create a policy map for the class, and apply the policy map to an interface for QoS.

Examples: Classification for voice and video

This example describes how to classify packet streams for voice and video using device specific information.

In this example, voice and video are coming in from end-point A into HundredGigabitEthernet1/0/1 on the device and have precedence values of 5 and 6, respectively. Additionally, voice and video are also coming from end-point B into HundredGigabitEthernet1/0/3 on the device with DSCP values of EF and AF11, respectively.

Assume that all the packets from the both the interfaces are sent on the uplink interface, and there is a requirement to police voice to 100 Mbps and video to 150 Mbps.

To classify per the above requirements, a class to match voice packets coming in on HundredGigabitEthernet1/0/1 is created, named voice-interface-1, which matches precedence 5. Similarly another class for voice is created, named voice-interface-2, which will match voice packets in HundredGigabitEthernet1/0/3. These classes are associated to two separate policies named input-interface-1, which is attached to HundredGigabitEthernet1/0/1, and input-interface- 2, which is attached to HundredGigabitEthernet1/0/3. The action for this class is to mark the QoS-group to 10. To match packets with QoS-group 10 on the output interface, a class named voice is created which matches on QoS-group 10. This is then associated to another policy named output-interface,which is associated to the uplink interface. Video is handled in the same way, but matches on QoS-group 20.

This example shows how classify using the above device specific information:

```
Device(config-cmap)# class-map voice-interface-1
Device(config-cmap)# match ip precedence 5
Device(config-cmap)# exit

Device(config)# class-map video-interface-1
Device(config-cmap)# match ip precedence 6
Device(config-cmap)# exit

Device(config)# class-map voice-interface-2
Device(config-cmap)# match ip dscp ef
Device(config-cmap)# exit

Device(config)# class-map video-interface-2
Device(config-cmap)# match ip dscp af11
Device(config-cmap)# exit

Device(config)# policy-map input-interface-1
Device(config-pmap)# class voice-interface-1
Device(config-pmap-c)# set qos-group 10
Device(config-pmap-c)# set traffic-class 7

Device(config-pmap-c)# class video-interface-1
Device(config-pmap-c)# set qos-group 20
Device(config-pmap-c)# set traffic-class 6

Device(config-pmap-c)# policy-map input-interface-2
Device(config-pmap)# class voice-interface-2
Device(config-pmap-c)# set qos-group 10
Device(config-pmap-c)# set traffic-class 7
Device(config-pmap-c)# class video-interface-2
Device(config-pmap-c)# set qos-group 20
Device(config-pmap-c)# set traffic-class 6

Device(config)# class-map match-all traffic-class7
Device(config-cmap)# match traffic-class 7
Device(config-cmap)# exit
```

```
Device(config)# class-map match-all traffic-class6
Device(config-cmap)# match traffic-class 6
Device(config-cmap)# exit

Device(config)# interface HundredGigE1/0/1
Device(config-if)# service-policy input voice-inteface-1
Device(config-if)# exit
Device(config)# interface HundredGigE1/0/3
Device(config-if)# service-policy input input-interface-2
Device(config-if)# exit
Device(config)# interface HundredGigE1/0/5
Device(config-if)# service-policy type queuing output output-interface
Device(config-if)# exit

Device(config)# policy-map type queueing llq
Device(config-pmap)# class traffic-class7
Device(config-pmap-c)# shape average 1g
Device(config-pmap-c)# priority level 1
Device(config-pmap-c)# class traffic-class6
Device(config-pmap-c)# shape average percent 10
Device(config-pmap-c)# class traffic-class5
Device(config-pmap-c)# shape average 1000000000
Device(config-pmap-c)# random-detect discard-class-based
Device(config-pmap-c)# random-detect exponential-weighting-constant 1
Device(config-pmap-c)# random-detect discard-class 0 percent 50 100 1
Device(config-pmap-c)# random-detect discard-class 1 percent 25 75 1
Device(config-pmap-c)# end
```

Examples: Queue-limit configuration

This example shows how to configure a queue-limit policy based on per traffic-class:

```
Device# configure terminal
Device#(config)# policy-map type queueing test
Device#(config-pmap)# class tc7
Device#(config-pmap)# priority level 1
Device#(config-pmap-c)# shape average 1000000000
Device#(config-pmap-c)# queue-limit 100000000 bytes
Device#(config-pmap-c)# exit

Device#(config-pmap)# class tc6
Device#(config-pmap-c)# shape average 2000000000
Device#(config-pmap-c)# queue-limit 2000000000 bytes
Device#(config-pmap-c)# exit

Device#(config-pmap)# class tc5
Device#(config-pmap-c)# shape average 3000000000
Device#(config-pmap-c)# queue-limit 3000000000 bytes
Device#(config-pmap-c)# exit

Device#(config-pmap)# class tc4
Device#(config-pmap-c)# shape average 4000000000
Device#(config-pmap-c)# queue-limit 100000000 bytes
Device#(config-pmap-c)# exit

Device#(config-pmap)# class tc3
Device#(config-pmap-c)# shape average 5000000000
Device#(config-pmap-c)# queue-limit 200000000 bytes
Device#(config-pmap-c)# exit
```

```
Device#(config-pmap)# class tc2
Device#(config-pmap-c)# shape average 4000000000
Device#(config-pmap-c)# queue-limit 300000000 bytes
Device#(config-pmap-c)# exit

Device#(config-pmap)# class tc1
Device#(config-pmap-c)# shape average 3000000000
Device#(config-pmap-c)# queue-limit 100000000 bytes
Device#(config-pmap-c)# exit

Device#(config-pmap)# class class-default
Device#(config-pmap-c)# shape average 2000000000
Device#(config-pmap-c)# queue-limit 100000000 bytes
Device#(config-pmap-c)# end
Device#
```

Examples: Policing action configuration

This example displays the various policing actions that can be associated to the policer. These actions are accomplished using the conforming, exceeding, or violating packet configurations. You have the flexibility to drop, mark and transmit, or transmit packets that have exceeded or violated a traffic profile.

For example, a common deployment scenario is one where the enterprise customer polices traffic exiting the network towards the service provider and marks the conforming, exceeding and violating packets with different DSCP values. The service provider could then choose to drop the packets marked with the exceeded and violated DSCP values under cases of congestion, but may choose to transmit them when bandwidth is available.

Note The Layer 2 fields can be marked toinclude the CoS fields, and the Layer 3 fields can be marked to include the precedence and the DSCP fields.

One useful feature is the ability to associate multiple actions with an event. For example, you can set the precedence bit and the CoS for all conforming packets. A submode for an action configuration could then be provided by the policing feature.

This is an example of a policing action configuration:

```
Device# configure terminal
Device(config)# policy-map police
Device(config-pmap)# class class-default
Device(config-pmap-c)# police cir 1000000 pir 2000000
Device(config-pmap-c-police)# conform-action transmit
Device(config-pmap-c-police)# exceed-action set-dscp-transmit dscp table exceed-markdown-table
Device(config-pmap-c-police)# violate-action set-dscp-transmit dscp table
violate-markdown-table
Device(config-pmap-c-police)# end
```

In this example, the exceed-markdown-table and violate-mark-down-table are table maps.

Note Policer-based markdown actions are only supported using table maps. Only one markdown table map is allowed for each marking field in the device.

Examples: Policing units

The policing unit is the basis on which the token bucket works. Burst parameters are specified in bytes by default and cannot be configured. CIR and PIR are specified in bits per second, and can also be configured in percent.

This is an example of a policer configuration in bits per second. In this configuration, a dual-rate three-color policer is configured where the units of measurement is bits. The burst and peak burst are all specified in bits.

```
Device(config)# policy-map bps-policer
Device(config-pmap)# class class-default
Device(config-pmap-c)# police rate 1200000 peak-rate 12000000 conform-action transmit
exceed-action set-dscp-transmit dscp table DSCP_EXCE violate-action drop
```

Examples: Single-rate two-color policing configuration

This example shows how to configure a single-rate two-color policer:

```
Device(config)# class-map match-any prec1
Device(config-cmap)# match ip precedence 1
Device(config-cmap)# exit
Device(config)# policy-map policer
Device(config-pmap)# class prec1
Device(config-pmap-c)# police cir 256000 conform-action transmit exceed-action drop
Device(config-pmap-c-police)# exit
Device(config-pmap-c)#
```

Examples: Dual-rate three-color policing configuration

This example shows how to configure a dual-rate three-color policer:

```
Device# configure terminal Device(config)# policy-map test
Device(config-pmap)# class class-default
Device(config-pmap-c)# police cir 1000000000 pir 2000000000
Device(config-pmap-c-police)# conform-action transmit
Device(config-pmap-c-police)# exceed-action set-dscp-transmit dscp table exceed-markdown-table
Device(config-pmap-c-police)# violate-action set-dscp-transmit dscp table
violate-markdown-table
Device(config-pmap-c-police)# exit
Device(config-pmap-c)#
```

In this example, the exceed-markdown-table and violate-mark-down-table are table maps.

Note Policer based markdown actions are only supported using table maps.

Examples: Table map marking configuration

These steps and examples show how to use table map marking for your QoS configuration:

1. Define the table-map using the **table-map** command and indicate the mapping of the values. This table does not know of the policies or classes within which it will be used. The default command in the table map indicates the value to be copied into the ‘to’ field when there is no matching ‘from’ field. In the

example, a table map named table-map1 is created. The mapping defined is to convert the value from 0 to 1 and from 2 to 3, while setting the default value to 4.

Note Only table-map values of same QoS tags are supported. For example, a table-map of type CoS to CoS or DSCP to DSCP or Prec to Prec are supported, and also a table-map of type DSCP/CoS/PREC to a QoS groupand a table-map of type DSCP/CoS/PREC to a traffic-class.

```
Device(config)# table-map table-map1
Device(config-tablemap)# map from 0 to 1
Device(config-tablemap)# map from 2 to 3
Device(config-tablemap)# default 4
Device(config-tablemap)# exit
```

2. Define the policy map where the table map will be used.

   In the example, the incoming CoS is mapped to the CoS based on the mapping specified in the table table-map1. For this example, if the incoming packet has a CoS of 0, the CoS in the packet is set 1. If no table map name is specified the command assumes a default behavior where the value is copied as is from the 'from' field to the 'to' field.

   ```
   Device(config)# policy-map policy1
   Device(config-pmap)# class class-default
   Device(config-pmap-c)# set cos cos table table-map1
   Device(config-pmap-c)# exit
   ```

3. Associate the policy to an interface.

   ```
   Device(config)# interface HundredGigE1/0/2
   Device(config-if)# service-policy output policy1
   Device(config-if)# exit
   ```

Display QoS configuration

This is a sample output of the **show policy-map** command:

```
Device# show policy-map mul4

Policy Map mul4
Class cs2
police cir 1000000000 bc 1024000
conform-action transmit
exceed-action drop
set dscp cs7
Class cs3
set traffic-class 7
police cir 40000000000 bc 1024000
conform-action transmit
exceed-action drop
Class cs1
police cir 1000000000 bc 1024000
conform-action transmit
exceed-action drop
set traffic-class 5
Class cs5
police cir 20000000000 bc 1024000
conform-action transmit
exceed-action drop
```

```
set traffic-class 5
Class cs6
police cir 50000000 bc 10240
conform-action transmit
exceed-action drop
Class dscp1
police cir 5500000000 bc 1024000
conform-action transmit
exceed-action drop
set traffic-class 2
set dscp 45
Class ttl
police cir 1000000000 bc 1024000
conform-action transmit
exceed-action drop
Class cs4
police cir 10000000000 bc 1024000
conform-action transmit
exceed-action drop
set traffic-class 5
Class class-default
police cir 1000000000
bc 1024000
conform-action transmit
exceed-action drop
```

This is a sample output of the **show policy-map interface** command:

```
Device# show policy-map interface FiftyGigE1/0/14

FiftyGigE1/0/14
Service-policy input: mul4

Class-map: cs2 (match-all)
0 packets
Match: dscp cs2 (16)
police:
cir 1000000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
QoS Set
dscp cs7

Class-map: cs3 (match-all)
224757946122 packets
Match: dscp cs3 (24)
QoS Set
traffic-class 7
police:
cir 40000000000 bps, bc 1024000 bytes
conformed 335092644777000 bytes; actions:
transmit
exceeded 2044274406000 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps

Class-map: cs1 (match-all)
0 packets
Match: dscp cs1 (8)
police:
cir 1000000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
```

```
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
QoS Set
traffic-class 5

Class-map: cs5 (match-all)
0 packets
Match: dscp cs5 (40)
police:
cir 20000000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
QoS Set
traffic-class 5

Class-map: cs6 (match-all)
0 packets
Match: dscp cs6 (48)
police:
cir 50000000 bps, bc 10240 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps

Class-map: dscp1 (match-all)
0 packets
Match: dscp 5
police:
cir 5500000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
QoS Set
traffic-class 2
dscp 45

Class-map: ttl (match-all)
0 packets
Match: access-group name ttl
police:
cir 1000000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps

Class-map: cs4 (match-all)
0 packets
Match: dscp cs4 (32)
police:
cir 10000000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
```

```
drop
conformed 0000 bps, exceeded 0000 bps
QoS Set
traffic-class 5

Class-map: class-default (match-any)
5073 packets
Match: any
police:
cir 1000000000 bps, bc 1024000 bytes
conformed 1215000 bytes; actions:
transmit
exceeded 6394500 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
```

These are sample outputs of the **show policy-map type queue interface** command. This command displays information about which ingress packets are hitting which VoQs.

Output from Cisco C9350 series smart switches:

```
Device# show policy-map type queue interface FiftyGigE1/0/14

 FiftyGigE1/0/14

  Service-policy queueing output: llq

    queue stats for all priority classes:
      Queueing
      priority level 1
      queue limit 96000 bytes
      (total drops) 0
      (bytes output) 59924103953524

    queue stats for all priority classes:
      Queueing
      priority level 2
      queue limit 96000 bytes
      (total drops) 0
      (bytes output) 0

    queue stats for all priority classes:
      Queueing
      priority level 3
      queue limit 96000 bytes
      (total drops) 0
      (bytes output) 0

    queue stats for all priority classes:
      Queueing
      priority level 4
      queue limit 96000 bytes
      (total drops) 0
      (bytes output) 0

    queue stats for all priority classes:
      Queueing
      priority level 5
      queue limit 96000 bytes
      (total drops) 0
      (bytes output) 0

    queue stats for all priority classes:
      Queueing
      priority level 6
```

```
  queue limit 96000 bytes
  (total drops) 0
  (bytes output) 0

queue stats for all priority classes:
  Queueing
  priority level 7
  queue limit 96000 bytes
  (total drops) 0
  (bytes output) 0

Class-map: tc7 (match-all)
  67482284685 packets
  Match: traffic-class 7
  shape (average) cir 5000000000, bc 20000000, be 20000000
  target shape rate 5000000000
  Priority: Strict,

  Priority Level: 1

Class-map: tc6 (match-all)
  0 packets
  Match: traffic-class 6
  shape (average) cir 1500000000, bc 6000000, be 6000000
  target shape rate 1500000000
  Priority: Strict,

  Priority Level: 2

Class-map: tc5 (match-all)
  0 packets
  Match: traffic-class 5
  Priority: Strict,

  Priority Level: 3
  shape (average) cir 2000000000, bc 8000000, be 8000000
  target shape rate 2000000000

Class-map: tc4 (match-all)
  0 packets
  Match: traffic-class 4
  Priority: Strict,

  Priority Level: 4
  shape (average) cir 1500000000, bc 6000000, be 6000000
  target shape rate 1500000000

Class-map: tc3 (match-all)
  0 packets
  Match: traffic-class 3
  Priority: Strict,

  Priority Level: 5
  shape (average) cir 1500000000, bc 6000000, be 6000000
  target shape rate 1500000000

Class-map: tc2 (match-all)
  0 packets
  Match: traffic-class 2
  Priority: Strict,

  Priority Level: 6
  shape (average) cir 1500000000, bc 6000000, be 6000000
  target shape rate 1500000000
```

```
    Class-map: tc1 (match-all)
      0 packets
      Match: traffic-class 1
      shape (average) cir 40000000000, bc 400000000, be 400000000
      target shape rate 40000000000
      Priority: Strict,

      Priority Level: 7

    Class-map: class-default (match-any)
      35230 packets
      Match: any
      Queueing
      queue limit 7500000 bytes
      (total drops) 0
      (bytes output) 0
      shape (average) cir 100000000, bc 400000, be 400000
      target shape rate 100000000
```

Output from Cisco C9610 series smart switches:

```
Device# do sh policy-map ty q interface FiftyGigE1/0/1
FiftyGigE1/0/1
Service-policy queueing output: llq
queue stats for all priority classes:
Queueing
priority level 1
queue limit 32256 bytes
(total drops) 0
(bytes output) 0
queue stats for all priority classes:
Queueing
priority level 2
queue limit 32256 bytes
(total drops) 0
(bytes output) 0
Class-map: tc7 (match-all)
0 packets
Match: traffic-class 7
Priority: Strict,

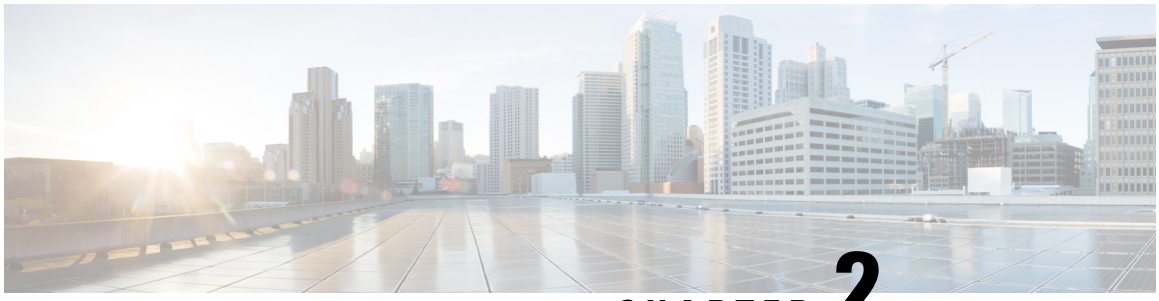
Priority Level: 1
shape (average) cir 10000000000, bc 40000000, be 40000000
target shape rate 10000000000
Class-map: tc6 (match-all)
0 packets
Match: traffic-class 6
shape (average) cir 1000000000, bc 10000000, be 10000000
target shape rate 1000000000
Priority: Strict,

Priority Level: 2
Class-map: tc5 (match-all)
0 packets
Match: traffic-class 5
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 250000000, bc 1000000, be 1000000
target shape rate 250000000
Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
```

```
thresh thresh prob
0 50 100 1/1
1 25 75 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: tc4 (match-all)
0 packets
Match: traffic-class 4
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 10000000000, bc 100000000, be 100000000
target shape rate 10000000000
Class-map: tc3 (match-all)
0 packets
Match: traffic-class 3
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 100000000, bc 1000000, be 1000000
target shape rate 100000000
Class-map: tc2 (match-all)
0 packets
Match: traffic-class 2
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 100000000000, bc 400000000, be 400000000
target shape rate 100000000000
Class-map: tc1 (match-all)
0 packets
Match: traffic-class 1
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 100000000000, bc 400000000, be 400000000
target shape rate 100000000000
bandwidth remaining ratio 10
Class-map: class-default (match-any)
0 packets
Match: any
Queueing
queue limit 1000000 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 1000000000, bc 4000000, be 4000000
target shape rate 1000000000
Exp-weight-constant: 3 (1/8)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 20 90 1/1
1 20 70 1/1
2 0 0 1/1
3 0 0 1/1
```

```
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
```



CHAPTER 2

Auto-QoS

This document provides detailed understanding about Auto-QoS and its configuration.

- Feature history for Auto-QoS, on page 81
- Auto-QoS overview, on page 82
- Auto-QoS compact overview, on page 82
- Limitations for Auto-QoS, on page 83
- How Auto-QoS generates QoS templates, on page 84
- Auto-QoS policy and class maps, on page 84
- Behavior when Auto-QoS is enabled, on page 85
- Behavior when Auto-QoS compact is enabled, on page 85
- How to configure Auto-QoS, on page 87
- Upgrade Auto-QoS, on page 88
- Enable Auto-QoS compact, on page 90
- Monitor Auto-QoS, on page 92
- Troubleshoot Auto-QoS, on page 92
- Configuration examples, on page 93

Feature history for Auto-QoS

This table provides release and platform support information for the features explained in this module.

These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Release	Feature name and description	Supported platform
Cisco IOS XE 26.2.1ea	Auto-QoS: Auto-QoS feature support has been introduced.	Cisco C9550 Series Smart Switches
Cisco IOS XE 17.18.1	Auto-QoS: Auto-QoS feature simplifies the deployment of QoS features. This feature determines the network design and enables QoS configurations so that the switch can prioritize different traffic flows.	Cisco C9350 Series Smart Switches

Auto-QoS overview

Auto-QoS is a feature designed to simplify the deployment of Quality of Service (QoS) on network devices. It automates the configuration of QoS policies by detecting the type of traffic (such as voice or video) and applying appropriate QoS settings without requiring manual, detailed configuration. This helps ensure that critical traffic receives the necessary priority and bandwidth for optimal performance.

Key points

- Automatic detection and configuration: Auto-QoS identifies network design and traffic types, then enables QoS configurations so the switch can prioritize different traffic flows.
- MQC model:
  - Uses the Modular QoS CLI (MQC) model.
  - Instead of individual global configurations, Auto-QoS applied to any interface generates several global class maps and policy maps.
- Traffic classification:
  - Auto-QoS classifies traffic and assigns each matched packet to a qos-group.
  - The output policy map places specific qos-groups into designated output queues, including the priority queue for real-time traffic.
- Egress queuing: Egress queuing policy maps traffic-class to output queues.
- Ingress policy: traffic-class must be set explicitly; otherwise, default mappings use only traffic-class 0 or 7.
- Trust behavior: DSCP is trusted by default for routed packets; CoS is trusted by default for bridged packets.
- Bi-directional QoS:
  - Inbound: Switch port should trust the DSCP value in the packet (default behavior).
  - Outbound: Switch port must ensure voice packets receive "front of line" priority, preventing excessive delay that could lead to packet loss at the end host.

Why is QoS important?

QoS ensures timely delivery and prioritization of critical traffic (like voice and video) by managing bandwidth and scheduling at both inbound and outbound interfaces.

Auto-QoS compact overview

Auto-QoS is a Cisco feature that automatically generates and applies recommended Quality of Service (QoS) configurations for your network devices. This simplifies QoS deployment by reducing the need for manual, detailed QoS design and configuration.

When you enter an Auto-QoS command, the switch displays all the generated commands as if they were entered manually via the CLI. This provides transparency into the QoS policies being applied.

Key benefits

- Simplifies configuration review: Reduces clutter in the running configuration.
- Efficient memory usage: Saves resources by not listing all auto-generated CLI lines.
- Easy deployment: Maintains the simplicity of Auto-QoS while enhancing manageability.

Limitations for Auto-QoS

These are restrictions for auto-QoS:

- Auto-qos is supported on Layer 2 or Layer 3 ports.
- Auto-QoS is not supported on subinterfaces and tunnel interfaces and member ports.
- Auto-QoS does not generate configuration when it is pushed from the startup-configuration using the auto qos voip cisco-phone command to the running-configuration. This is expected behavior and this is to prevent overwriting of user-created customized QoS policies by the default configuration, if any, every time the command auto qos voip cisco-phone is pushed from the startup-config.

  Use any of these workarounds for this limitation:

  - Configure the auto qos voip cisco-phone command manually on the switch interfaces.
  - For new switches, if you push auto-QoS commands through startup-config, the command should include each of these as part of the standard template:
    - Interface-level:
      - trust device cisco-phone
      - auto qos voip cisco-phone
      - service-policy input AutoQos-4.0-CiscoPhone-Input-Policy
      - service-policy type queueing output AutoQos-4.0-Output-Policy
    - Global-level:
      - Class-map
      - Policy-map
      - ACL (ACE)
  - If the **auto qos voip cisco-phone** command is already configured on an interface but policies are not being generated, disable the command from all the interfaces and reconfigure the command on each interface manually.

How Auto-QoS generates QoS templates

- Class map generation: An Auto-QoS command automatically creates a series of class maps that match on Access Control Lists (ACLs), DSCP values, and/or CoS values. This differentiates network traffic into various application classes (such as voice, video, and data).
- Input policy generation: Auto-QoS also generates an input policy that references these class maps. In some cases, this input policy includes policing (rate-limiting) for specific classes to enforce bandwidth limits.
- Egress queue class maps: Auto-QoS generates eight egress-queue class maps. The egress (output) policy then assigns each of these class maps to a specific output queue, ensuring appropriate queueing and scheduling behavior for each traffic type.

Template and resource efficiency

- Template generation: The first time an Auto-QoS command is used on a device, it creates global templates—such as the eight-queue egress service-policy—that define the basic QoS structure.
- Template reuse: After these templates are created, subsequent Auto-QoS commands applied to other interfaces do not re-create them. All Auto-QoS-enabled interfaces share the same global queueing model and templates, ensuring consistency and efficient use of device resources.

Key points

- Auto-QoS simplifies traffic classification and queue assignment by generating standard templates and policies.
- Templates are generated once and reused, optimizing configuration and memory usage across the device.
- Eight-queue model is central to Cisco's Auto-QoS approach, covering a broad range of application classes with standardized output queueing.

Auto-QoS policy and class maps

When you enter the appropriate Auto-QoS command, these actions occur automatically:

- Creation of specific class maps: Auto-QoS generates class maps that define how different types of traffic (such as voice, video, and data) are identified, often based on DSCP or CoS values.
- Creation of specific policy maps (input and output): Corresponding policy maps are created for both inbound (input) and outbound (output) traffic, referencing the generated class maps and defining actions such as marking, policing, or queuing.
- Attachment of policy maps to the interface: The generated policy maps are automatically applied (attached) to the specified interface where you enabled Auto-QoS.
- Configuration of the interface trust level: The trust state for the interface is set, determining whether the switch will accept and honor incoming QoS markings (such as DSCP or CoS) from connected devices.
- DSCP is trusted by default for routed packets, and CoS is trusted by default for bridged packets.

Behavior when Auto-QoS is enabled

- Addition of commands: When Auto-QoS is enabled, both the auto qos interface commands and the globally generated QoS configuration are automatically added to the device's running configuration.
- Command application: The switch applies the Auto-QoS-generated commands as if they were manually entered via the CLI.
- Interaction with existing configuration:
  - If there are existing user configurations, Auto-QoS-generated commands may fail to apply or may override user settings—sometimes without warning.
  - If all generated commands are applied successfully, any user configuration not overridden remains in the running configuration.
  - Any user configuration that was overridden can be restored by reloading the switch without saving the current (modified) configuration.
- Rollback behavior: If Auto-QoS-generated commands cannot be applied, the switch restores the previous running configuration.

Key points

- Enabling Auto-QoS modifies the running configuration and may override user-entered settings.
- Overridden user configuration is not permanently lost unless you save the running configuration after enabling Auto-QoS.
- Safe rollback: If Auto-QoS configuration fails, your previous running config is restored automatically.

Behavior when Auto-QoS compact is enabled

- Running configuration:

Note When you enable Auto-QoS on the main interface, the **show running-config interface <name>** command does not display meter and queuing policy information.

  - Only the auto-qos commands entered by the user are visible in the running configuration.
  - The detailed, auto-generated global and interface QoS configurations are hidden from the running-config output.
  - The auto qos global compact command hides derived configurations from the running configuration, preventing Auto-QoS-generated commands from appearing in the output. In this instance, the no auto qos global compact command failed to disable the feature, which kept interface service-policy commands hidden and created the appearance that auto qos trust dscp was not functioning as expected.

- You cannot directly disable the auto qos global compact command while Auto-QoS configurations, such as auto qos trust dscp, remain active on an interface. Attempting to do so triggers the following error message:

```
Device(config)#no auto qos global compact
%Disable all the autoqos configuration from the interfaces to disable autoqos compact
```

- To disable auto qos global compact, first remove all Auto-QoS configurations from the interfaces, then disable the global command.

- Configuration save/reload:
  - When you save the configuration, only the auto-qos commands are stored (not the hidden, generated configuration).
  - Upon reload, the switch detects the saved auto-qos commands and re-executes them, regenerating the full Auto-QoS configuration (using the latest SRND-compliant config-set).

Important considerations

No manual edits: Do not make changes to the hidden, auto-generated commands while Auto-QoS Compact is enabled. Any user modifications will be lost (overridden) when the switch reloads and the configuration is regenerated.

Auto-QoS global compact mode

- Viewing derived config: Use the show derived-config command to view the hidden Auto-QoS global compact configuration.
- Memory and regeneration: Auto-QoS global compact-derived commands are not stored in memory. They are regenerated on every reload based on the current auto-qos commands present in the saved configuration.
- Modification warning: When compaction is enabled, do not modify auto-generated commands directly; any such changes will be lost at reload.
- Disabling compact: If you need to disable Auto-QoS global compact, first disable Auto-QoS at the interface level before disabling compact mode at the global level.

Table showing comparison of standard and compact Auto-QoS

Feature/action	Standard Auto-QoS	Compact Auto-QoS
User CLI commands in config	Visible	Visible
Auto-generated config	Visible	Hidden
Save config	All commands	Only auto-qos CLI commands
Reload behavior	Normal reload	Auto-QoS config regenerated
Manual edits to auto-gen config	Allowed	Not recommended (overwritten)

How to configure Auto-QoS

For optimum QoS performance, configure auto-QoS on all the devices in your network.

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 2	**interface** *interface-id* **Example:** `Device(config)# interface FiftyGigE1/0/1`	Specifies the port connected to a VoIP phone, video device, or uplink. Enters interface configuration mode.
Step 3	Depending on your auto-QoS configuration, use one of the following commands: • **auto qos voip** {**cisco-phone** | **cisco-softphone** | **trust**} • **auto qos video** {**cts** | **ip-camera** | **media-player**} • **auto qos classify** [**police**] • **auto qos trust** {**cos** | **dscp**} **Example:** `Device(config-if)# auto qos trust dscp`	Enables Auto-QoS for the interface and device type. See details and notes below: • **auto qos voip cisco-phone**: For Cisco IP Phones. Trusts QoS labels when the phone is detected via CDP. **Note** Do not use for IP phones that support video, as video DSCP markings may be overwritten and misclassified. • **auto qos voip cisco-softphone**: For PCs with Cisco IP SoftPhone. Generates QoS config and policing for untrusted endpoints. • **auto qos voip trust**: For uplinks to trusted switches or routers. Trusts VoIP QoS classification on ingress packets. These commands enable auto-QoS for the specified video device (system, camera, or media player): • **auto qos video cts**: For Cisco TelePresence systems. Trusts QoS labels when detected.

	Command or Action	Purpose
		• **auto qos video ip-camera**: For Cisco video surveillance cameras. Trusts QoS labels when detected. • **auto qos video media-player**: For Cisco digital media players. Trusts QoS labels when detected. This command enables auto-QoS for classification: • **auto qos classify police**: For untrusted interfaces. Classifies and marks traffic, applies policing. These commands enable auto-QoS for trusted interfaces: • **auto qos trust cos**/**auto qos trust dscp**: Trusts CoS or DSCP values, typically for uplink or trunk ports.
Step 4	**end** **Example:** `Device(config-if)# end`	Returns to privileged EXEC mode.
Step 5	**show auto qos interface** *interface-id* **Example:** `Device# show auto qos interface FiftyGigE1/0/1`	(Optional) Displays the auto-QoS command on the interface on which auto-QoS was enabled. Use the **show running-config** command to display the auto-QoS configuration and user modifications.

Upgrade Auto-QoS

Before you begin

Prior to upgrading, you need to remove all auto-QoS configurations currently on the switch. This sample procedure describes that process.

After following this sample procedure, you must then reboot the switch with the new or upgraded software image and reconfigure auto-QoS.

Procedure

	Command or Action	Purpose

Step 1	**show auto qos**
	Example:
	`Device# show auto qos`
	`TwentyFiveGigE1/0/1`
	`auto qos trust dscp`
	`TwentyFiveGigE1/0/2`
	`auto qos trust cos`
		In privileged EXEC mode, record all current auto QoS configurations by entering this command.

Step 2	**no auto qos** **Example:** `Device(config-if)# no auto qos`	**In interface configuration mode**, run the appropriate **no auto qos** command on each interface that has an auto QoS configuration.

Step 3	**show running-config | i autoQos**
	Example:
	`Device# show running-config | i autoQos`
	`Device (config-if)#do sh ru | in auto qos`
	`auto qos trust dscp`
	`auto qos trust cos`
	`auto qos trust`
		Return to privileged EXEC mode, and record any remaining auto QoS maps class maps, policy maps, access lists, table maps, or other configurations by entering this command.

Step 4	**no policy-map** *policy-map_name*
	Example:
	`Device(config)# no policy-map pmap_101`
	`Device(config)# no class-map cmap_101`
	`Device(config)# no ip access-list extended AutoQos-101`
	`Device(config)# no table-map 101`
	`Device(config)# no table-map policed-dscp`
		In global configuration mode, remove the QoS class maps, policy maps, access-lists, table maps, and any other auto QoS configurations by entering these commands:
		• **no policy-map** *policy-map-name*
		• **no class-map** *class-map-name*
		• **no ip access-list extended** *Auto-QoS-x*
		• **no table-map** *table-map-name*
		• **no table-map policed-dscp**

Step 5	**show running-config | i autoQos**
	Example:
	`Device# sh ru | in AutoQos`
	`Device# (config-if)#do sh ru | in AutoQos`
	`auto qos trust cos`
	`auto qos video cts`
		Return to privileged EXEC mode, run this command again to ensure that no auto-QoS configuration or remaining parts of the auto-QoS configuration exists

	Command or Action	Purpose

	`auto qos trust dscp`
	`auto qos video media-player`
	`auto qos trust cos`
	`auto qos video ip-camera`	

Step 6	**show auto qos** **Example:** `Device# show auto qos`	Run this command to ensure that no auto-QoS configuration or remaining parts of the configuration exists.
Step 7	**write memory** **Example:** `Device# write memory`	Write the changes to the auto QoS configuration to NV memory by entering the **write memory** command.

What to do next

Reboot the switch with the new or upgraded software image.

After rebooting with the new or upgraded software image, re-configure auto-QoS for the appropriate switch interfaces as determined by running the **show auto qos** command described in step 1.

Note On a Cisco switch or switch stack, only one table-map can be applied for the exceed action and one for the violate action (markdown) per device.

If a table-map is already in use under the exceed action, attempting to apply an Auto-QoS policy that also requires a table-map for the exceed action will fail.

Enable Auto-QoS compact

To enable auto-QoS compact, enter these commands:

Procedure

	Command or Action	Purpose
Step 1	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 2	**auto qos global compact** **Example:**	Enables auto-Qos compact and generates (hidden) the global configurations for auto-QoS.

	Command or Action	Purpose
	`Device(config)# auto qos global compact`	You can then enter the auto-QoS command you want to configure in the interface configuration mode and the interface commands that the system generates are also hidden. To display the auto-QoS configuration that has been applied, use these the privileged EXEC commands: • **show derived-config** • **show policy-map** • **show access-list** • **show class-map** • **show table-map** • **show auto qos** • **show auto qos intefaces** • **show auto qos interface <interface name> FiftyGigE1/0/1** • **auto qos trust** • **show policy-map interface** • **show ip access-lists** • **show policy-map type queue** • **show policy-map type queueing interface** • **show table-map** These commands will have keyword "**AutoQos-**".

What to do next

To disable auto-QoS compact, remove auto-Qos instances from all interfaces by entering the **no** form of the corresponding auto-QoS commands and then enter the **no auto qos global compact** global configuration command.

Monitor Auto-QoS

Table 10: Commands for monitoring Auto-QoS

Command	Description
show auto qos [**interface** [*interface-id*]]	Displays the initial auto-QoS configuration. You can compare the **show auto qos** and the **show running-config** command output to identify the user-defined QoS settings.
show running-config	Displays information about the QoS configuration that might be affected by auto-QoS. You can compare the s**show auto qos** and the **show running-config** command output to identify the user-defined QoS settings.
show derived-config	Displays the hidden **mls qos** command which get configured along with the running configs because of auto-qos template.

Troubleshoot Auto-QoS

To troubleshoot Auto-QoS processes and issues:

- Use the **debug auto qos** privileged EXEC command.
- Enables detailed debugging output for Auto-QoS operations.
- Refer to the device's command reference for more details on usage and output interpretation.

To disable Auto-QoS on a specific port:

- Use the no form of the **interface** *interface-id* and **no auto qos voip** Auto-QoS interface commands.
- This removes only the Auto-QoS-generated interface configuration commands from the specified port.
- Global config behavior: If this is the last port with Auto-QoS enabled, entering the **no auto qos** command disables Auto-QoS on the device.

Note The global Auto-QoS-generated configuration commands remain in place. This avoids interrupting traffic on other ports that might still rely on those global settings.

Configuration examples

Example: auto qos trust cos

This is an example of the **auto qos trust cos** command and the applied policies and class maps.

These policy maps are created and applied when running this command:

- AutoQos-4.0-Trust-Cos-Input-Policy
- AutoQos-4.0-Output-Policy

These class maps are created and applied when running this command: **do sh policy-map ty q int fif 1/0/1 | in Class-map**

- AutoQos-4.0-Output-Priority-Queue (match-any)
- AutoQos-4.0-Output-Signaling-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
- AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
- AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
- AutoQos-4.0-Output-Scavenger-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
- class-default (match-any)

The **show policy-map interface** will only display the ingress policy information.

```
Device(config)# interface FiftyGigE1/0/1
        Device(config-if)# auto qos trust cos
        Device(config-if)# end

        Device# show policy-map interface FiftyGigE1/0/1

        FiftyGigE1/0/1

        Service-policy input: AutoQos-4.0-Trust-Cos-Input-Policy

        Class-map: class-default (match-any)
        0 packets
        Match: any
        QoS Set
        traffic-class cos table AutoQos-4.0-Trust-Cos-Table
```

To view the egress queue information (output policy), use the **show policy-map type queueing interface** command.

```
Device(config)# interface FiftyGigE1/0/1
        Device(config-if)# auto qos trust cos
        Device(config-if)# end

        Device# show policy-map type queueing interface FiftyGigE1/0/1
```

```
FiftyGigE1/0/1

Service-policy queueing output: AutoQos-4.0-Output-Policy

queue stats for all priority classes:
Queueing
priority level 1
queue limit 96000 bytes
(total drops) 0
(bytes output) 381284209

Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
3562992 packets
Match: traffic-class 7
Priority: Strict,

Priority Level: 1
shape (average) cir 12000000000, bc 120000000, be 120000000
target shape rate 12000000000

Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
0 packets
Match: traffic-class 6
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
0 packets
Match: traffic-class 5
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
```

```
Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
0 packets
Match: traffic-class 4
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
0 packets
Match: traffic-class 3
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 4

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
0 packets
Match: traffic-class 2
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 1

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
```

```
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
0 packets
Match: traffic-class 1
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: class-default (match-any)
3547910 packets
Match: any
Queueing
queue limit 75000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 23

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
```

Example: auto qos trust dscp

This is an example of the **auto qos trust dscp** command and the applied policies and class maps.

These policy maps are created and applied when running this command:

- AutoQos-4.0-Trust-Dscp-Input-Policy
- AutoQos-4.0-Output-Policy

These class maps are created and applied when running this command: **do sh policy-map ty q int FiftyGigE1/0/1 | in Class-map**

- class-default (match-any)

- AutoQos-4.0-Output-Priority-Queue (match-any)
- AutoQos-4.0-Output-Signaling-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
- AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
- AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
- AutoQos-4.0-Output-Scavenger-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
- class-default (match-any)
- show policy-map interface FiftyGigE1/0/1 | in Class-map
- AutoQos-4.0-Trust-Dscp-Priority-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Priority-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Signaling-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Signaling-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Transaction-Data-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Transaction-Data-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Bulk-Data-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Bulk-Data-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Scavenger-Class (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class2 (match-any)

The **show policy-map interface** command displays the ingress policy information

```
Device(config)# interface FiftyGigE1/0/1
Device(config-if)# auto qos trust dscp
Device(config-if)# end

Device# show policy-map interface FiftyGigE1/0/1

Service-policy input: AutoQos-4.0-Trust-Dscp-Input-Policy

Class-map: AutoQos-4.0-Trust-Dscp-Priority-Class1 (match-any)
0 packets
Match: dscp ef (46)
QoS Set
traffic-class 7

Class-map: AutoQos-4.0-Trust-Dscp-Priority-Class2 (match-any)
0 packets
Match: dscp cs4 (32) cs5 (40)
```

```
QoS Set
traffic-class 7
discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Signaling-Class1 (match-any)
0 packets
Match: dscp cs7 (56)
QoS Set
traffic-class 6

Class-map: AutoQos-4.0-Trust-Dscp-Signaling-Class2 (match-any)
0 packets
Match: dscp cs2 (16) cs3 (24) cs6 (48)
QoS Set
traffic-class 6
discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class1 (match-any)
0 packets
Match: dscp af41 (34)
QoS Set
traffic-class 5

Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class2 (match-any)
0 packets
Match: dscp af42 (36) af43 (38)
QoS Set
traffic-class 5
discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Transaction-Data-Class1 (match-any)
0 packets
Match: dscp af21 (18)
QoS Set
traffic-class 4

Class-map: AutoQos-4.0-Trust-Dscp-Transaction-Data-Class2 (match-any)
0 packets
Match: dscp af22 (20) af23 (22)
QoS Set
traffic-class 4
discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Bulk-Data-Class1 (match-any)
0 packets
Match: dscp af11 (10)
QoS Set
traffic-class 3

Class-map: AutoQos-4.0-Trust-Dscp-Bulk-Data-Class2 (match-any)
0 packets
Match: dscp af12 (12) af13 (14)
QoS Set
traffic-class 3
discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Scavenger-Class (match-any)
0 packets
Match: dscp cs1 (8)
QoS Set
traffic-class 2

Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class1 (match-any)
0 packets
```

```
Match: dscp af31 (26)
QoS Set
traffic-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class2 (match-any)
0 packets
Match: dscp af32 (28) af33 (30)
QoS Set
traffic-class 1
discard-class 1

Class-map: class-default (match-any)
0 packets
Match: any
```

To view egress queue information (output policy), use the **show policy-map type queueing interface** command:

```
Device(config)# interface FiftyGigE1/0/1
Device(config-if)# auto qos trust dscp
Device(config-if)# end

Device# show policy-map type queueing interface FiftyGigE1/0/1
Service-policy queueing output: AutoQos-4.0-Output-Policy
```

Note The queue limit for outbound traffic varies between XL and non-XL platforms. For non-XL platforms, the queue limit is typically configured based on time or bytes.

```
queue stats for all priority classes:
Queueing
priority level 1
queue limit 32256 bytes
(total drops) 0
(bytes output) 4284

Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
49 packets
Match: traffic-class 7
Priority: Strict,

Priority Level: 1
shape (average) cir 30000000000, bc 300000000, be 300000000
target shape rate 30000000000

Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
0 packets
Match: traffic-class 6
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
```

```
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
0 packets
Match: traffic-class 5
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
0 packets
Match: traffic-class 4
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
0 packets
Match: traffic-class 3
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 4

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
```

```
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
0 packets
Match: traffic-class 2
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 1

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
0 packets
Match: traffic-class 1
Queueing
queue limit 30000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: class-default (match-any)
50 packets
Match: any
Queueing
queue limit 75000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 23

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
```

```
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
```

Example: auto qos video cts

This is an example of the **auto qos video cts** command and the applied policies and class maps.

These policy maps are created and applied when running this command:

- AutoQos-4.0-Trust-Cos-Input-Policy
- AutoQos-4.0-Output-Policy

These class maps are created and applied when running this command: **sh policy-map ty q int FiftyGigE1/0/1 | in Class-map**

```
Device#sh policy-map ty q int FiftyGigE1/0/1 | in Class-map:
        Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
        Class-map: class-default (match-any)


        Device(config)# interface FiftyGigE1/0/1
        Device(config-if)# auto qos video cts
        Device(config-if)# end

        Device(config-if)# do sh policy-map interface FiftyGigE1/0/30
        FiftyGigE1/0/30

        Service-policy input: AutoQos-4.0-Trust-Cos-Input-Policy

        Class-map: class-default (match-any)
        0 packets
        Match: any
        QoS Set
        traffic-class cos table AutoQos-4.0-Trust-Cos-Table
        police:
        cir 5000000000 bps, bc 1024000 bytes
        conformed 0 bytes; actions:
        transmit
        exceeded 0 bytes; actions:
        drop
        conformed 0000 bps, exceeded 0000 bps
```

Queueing output

```
sh policy-map ty q inter fif 1/0/30
        FiftyGigE1/0/30

        Service-policy queueing output: AutoQos-4.0-Output-Policy
```

```
queue stats for all priority classes:
Queueing
priority level 1
queue limit 32256 bytes
(total drops) 0
(bytes output) 7267

Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
51 packets
Match: traffic-class 7
Priority: Strict,

Priority Level: 1
shape (average) cir 3000000000, bc 30000000, be 30000000
target shape rate 3000000000

Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
0 packets
Match: traffic-class 6
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 1000000000, bc 4000000, be 4000000
target shape rate 1000000000
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
0 packets
Match: traffic-class 5
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
```

```
0 packets
Match: traffic-class 4
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
0 packets
Match: traffic-class 3
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 4

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
0 packets
Match: traffic-class 2
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 1

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1
```

```
Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
0 packets
Match: traffic-class 1
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: class-default (match-any)
51 packets
Match: any
Queueing
queue limit 5000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 23

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1
```

Example: auto-qos video ip-camera

This is an example of the **auto qos video ip-camera** command and the applied policies and class maps.

These policy maps are created and applied when running this command:

- AutoQos-4.0-Trust-Dscp-Input-Policy
- AutoQos-4.0-Output-Policy

These class maps are created and applied when running this command:

```
Device(config-if)#auto qos video ip-camera
Device(config-if)#do sh policy-map int fif 1/0/1 | in auto
```

- AutoQos-4.0-Trust-Dscp-Input-Policy
- AutoQos-4.0-Trust-Dscp-Priority-Class1 (match-any)

- AutoQos-4.0-Trust-Dscp-Priority-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Signaling-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Signaling-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Transaction-Data-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Transaction-Data-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Bulk-Data-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Bulk-Data-Class2 (match-any)
- AutoQos-4.0-Trust-Dscp-Scavenger-Class (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class1 (match-any)
- AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class2 (match-any)
- Class-map: class-default (match-any)

```
Device(config)# interface FiftyGigE1/0/1
Device(config-if)# auto qos video ip-camera
Device(config-if)# end

Device(config-if)#do sh policy-map int FiftyGigE1/0/1
  FiftyGigE1/0/1

    Service-policy input: AutoQos-4.0-Trust-Dscp-Input-Policy

    Class-map: AutoQos-4.0-Trust-Dscp-Priority-Class1 (match-any)
      0 packets
        Match:  dscp ef (46)
        QoS Set
        traffic-class 7

    Class-map: AutoQos-4.0-Trust-Dscp-Priority-Class2 (match-any)
      0 packets
        Match:  dscp cs4 (32) cs5 (40)
        QoS Set
        traffic-class 7
        discard-class 1

    Class-map: AutoQos-4.0-Trust-Dscp-Signaling-Class1 (match-any)
      0 packets
        Match:  dscp cs7 (56)
        QoS Set
        traffic-class 6

    Class-map: AutoQos-4.0-Trust-Dscp-Signaling-Class2 (match-any)
      0 packets
        Match:  dscp cs2 (16) cs3 (24) cs6 (48)
        QoS Set
        traffic-class 6
        discard-class 1

    Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class1 (match-any)
```

```
  0 packets
    Match:  dscp af41 (34)
    QoS Set
    traffic-class 5

Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class2 (match-any)
  0 packets
    Match:  dscp af42 (36) af43 (38)
    QoS Set
    traffic-class 5
    discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Transaction-Data-Class1 (match-any)
  0 packets
    Match:  dscp af21 (18)
    QoS Set
    traffic-class 4

Class-map: AutoQos-4.0-Trust-Dscp-Transaction-Data-Class2 (match-any)
  0 packets
    Match:  dscp af22 (20) af23 (22)
    QoS Set
    traffic-class 4
    discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Bulk-Data-Class1 (match-any)
  0 packets
    Match:  dscp af11 (10)
    QoS Set
    traffic-class 3

Class-map: AutoQos-4.0-Trust-Dscp-Bulk-Data-Class2 (match-any)
  0 packets
   Match:  dscp af12 (12) af13 (14)
   QoS Set
   traffic-class 3
   discard-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Scavenger-Class (match-any)
  0 packets
    Match:  dscp cs1 (8)
    QoS Set
    traffic-class 2

Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class1 (match-any)
  0 packets
    Match:  dscp af31 (26)
    QoS Set
    traffic-class 1

Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class2 (match-any)
  0 packets
  Match:  dscp af32 (28) af33 (30)
  QoS Set
  traffic-class 1
  discard-class 1

Class-map: class-default (match-any)
  0 packets
    Match: any
    Device(config-if)#
```

This is a sample output of the **sh policy-map type queueing interface** command.

```
    Device(config-if)#do sh policy-map type queueing interface FiftyGigE1/0/1
    FiftyGigE1/0/1

    Service-policy queueing output: AutoQos-4.0-Output-Policy

    queue stats for all priority classes:
    Queueing
    priority level 1
    queue limit 32256 bytes
    (total drops) 0
    (bytes output) 5308

Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
  36 packets
    Match: traffic-class 7
    Priority: Strict,

    Priority Level: 1
    shape (average) cir 3000000000, bc 30000000, be 30000000
    target shape rate 3000000000

Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
  0 packets
    Match: traffic-class 6
    Queueing
    queue limit 300000 bytes
    (total drops) 0
    (bytes output) 0
    shape (average) cir 1000000000, bc 4000000, be 4000000
    target shape rate 1000000000
    bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
      Mean queue depth: 0
        Minimum Maximum  Mark
          thresh  thresh  prob
          0              85     100  1/1
          1              65     100  1/1
          2               0       0  1/1
          3               0       0  1/1
          4               0       0  1/1
          5               0       0  1/1
          6               0       0  1/1
          7               0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
  0 packets
    Match: traffic-class 5
    Queueing
    queue limit 300000 bytes
    (total drops) 0
    (bytes output) 0
    bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
      Mean queue depth: 0
        Minimum Maximum  Mark
          thresh  thresh  prob
          0              85     100  1/1
          1              65     100  1/1
          2               0       0  1/1
          3               0       0  1/1
          4               0       0  1/1
```

```
          5               0       0  1/1
          6               0       0  1/1
          7               0       0  1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
  0 packets
    Match: traffic-class 4
    Queueing
    queue limit 300000 bytes
    (total drops) 0
    (bytes output) 0
    bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
      Mean queue depth: 0
        Minimum Maximum  Mark
          thresh  thresh  prob
          0              85     100  1/1
          1              65     100  1/1
          2               0       0  1/1
          3               0       0  1/1
          4               0       0  1/1
          5               0       0  1/1
          6               0       0  1/1
          7               0       0  1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
  0 packets
    Match: traffic-class 3
    Queueing
    queue limit 300000 bytes
    (total drops) 0
    (bytes output) 0
    bandwidth remaining ratio 4

    Exp-weight-constant: 1 (1/2)
      Mean queue depth: 0
        Minimum Maximum  Mark
          thresh  thresh  prob
          0              85     100  1/1
          1              65     100  1/1
          2               0       0  1/1
          3               0       0  1/1
          4               0       0  1/1
          5               0       0  1/1
          6               0       0  1/1
          7               0       0  1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
  0 packets
    Match: traffic-class 2
    Queueing
    queue limit 300000 bytes
    (total drops) 0
    (bytes output) 0
    bandwidth remaining ratio 1

    Exp-weight-constant: 1 (1/2)
        Mean queue depth: 0
        Minimum Maximum  Mark
          thresh  thresh  prob
          0              85     100  1/1
          1              65     100  1/1
          2               0       0  1/1
```

```
            3               0       0  1/1
            4               0       0  1/1
            5               0       0  1/1
            6               0       0  1/1
            7               0       0  1/1

    Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
      0 packets
        Match: traffic-class 1
        Queueing
        queue limit 300000 bytes
        (total drops) 0
        (bytes output) 0
        bandwidth remaining ratio 9

        Exp-weight-constant: 1 (1/2)
            Mean queue depth: 0
            Minimum Maximum  Mark
              thresh  thresh  prob
              0               85     100  1/1
              1               65     100  1/1
              2                0       0  1/1
              3                0       0  1/1
              4                0       0  1/1
              5                0       0  1/1
              6                0       0  1/1
              7                0       0  1/1

    Class-map: class-default (match-any)
      36 packets
        Match: any
        Queueing
        queue limit 5000000 bytes
        (total drops) 0
        (bytes output) 0
        bandwidth remaining ratio 23

        Exp-weight-constant: 1 (1/2)
            Mean queue depth: 0
            Minimum Maximum  Mark
              thresh  thresh  prob
              0               85     100  1/1
              1               65     100  1/1
              2                0       0  1/1
              3                0       0  1/1
              4                0       0  1/1
              5                0       0  1/1
              6                0       0  1/1
              7                0       0  1/1
```

Example: auto qos video media-player

This is an example of the **auto qos video media-player** command and the applied policies and class maps.

These policy maps are created and applied when running this command:

- AutoQos-4.0-Trust-Dscp-Input-Policy
- AutoQos-4.0-Output-Policy

These class maps are created and applied when running this command: **do sh policy-map ty q interface FiftyGigE1/0/1 | in Class-map**

```
Device(config-if)#do sh policy-map ty q interface FiftyGigE1/0/1 | in Class-map:
        Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
        Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
        Class-map: class-default (match-any)


        Device(config)# interface FiftyGigE1/0/1
        Device(config-if)# auto qos video media-player
        Device(config-if)# end

        Device(config-if)#do sh policy-map interface FiftyGigE1/0/1
        FiftyGigE1/0/1

        Service-policy input: AutoQos-4.0-Trust-Dscp-Input-Policy

        Class-map: AutoQos-4.0-Trust-Dscp-Priority-Class1 (match-any)
        0 packets
        Match:  dscp ef (46)
        QoS Set
        traffic-class 7

        Class-map: AutoQos-4.0-Trust-Dscp-Priority-Class2 (match-any)
        0 packets
        Match:  dscp cs4 (32) cs5 (40)
        QoS Set
        traffic-class 7
        discard-class 1

        Class-map: AutoQos-4.0-Trust-Dscp-Signaling-Class1 (match-any)
        0 packets
        Match:  dscp cs7 (56)
        QoS Set
        traffic-class 6

        Class-map: AutoQos-4.0-Trust-Dscp-Signaling-Class2 (match-any)
        0 packets
        Match:  dscp cs2 (16) cs3 (24) cs6 (48)
        QoS Set
        traffic-class 6
        discard-class 1

        Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class1 (match-any)
        0 packets
        Match:  dscp af41 (34)
        QoS Set
        traffic-class 5

        Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Conf-Class2 (match-any)
        0 packets
        Match:  dscp af42 (36) af43 (38)
        QoS Set
        traffic-class 5
        discard-class 1

        Class-map: AutoQos-4.0-Trust-Dscp-Transaction-Data-Class1 (match-any)
        0 packets
        Match:  dscp af21 (18)
        QoS Set
        traffic-class 4
```

```
        Class-map: AutoQos-4.0-Trust-Dscp-Transaction-Data-Class2 (match-any)
        0 packets
        Match:  dscp af22 (20) af23 (22)
        QoS Set
        traffic-class 4
        discard-class 1

        Class-map: AutoQos-4.0-Trust-Dscp-Bulk-Data-Class1 (match-any)
        0 packets
        Match:  dscp af11 (10)
        QoS Set
        traffic-class 3

        Class-map: AutoQos-4.0-Trust-Dscp-Bulk-Data-Class2 (match-any)
        0 packets
        Match:  dscp af12 (12) af13 (14)
        QoS Set
        traffic-class 3
        discard-class 1

        Class-map: AutoQos-4.0-Trust-Dscp-Scavenger-Class (match-any)
        0 packets
        Match:  dscp cs1 (8)
        QoS Set
        traffic-class 2

        Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class1 (match-any)
        0 packets
        Match:  dscp af31 (26)
        QoS Set
        traffic-class 1

        Class-map: AutoQos-4.0-Trust-Dscp-Multimedia-Stream-Class2 (match-any)
        0 packets
        Match:  dscp af32 (28) af33 (30)
        QoS Set
        traffic-class 1
        discard-class 1

        Class-map: class-default (match-any)
        0 packets
        Match: any
```

Queueing output

```
Device(config-if)#do sh policy-map ty q int FiftyGigE1/0/30
 FiftyGigE1/0/30

  Service-policy queueing output: AutoQos-4.0-Output-Policy

    queue stats for all priority classes:
      Queueing
      priority level 1
      queue limit 32256 bytes
      (total drops) 0
      (bytes output) 5737

    Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
      25 packets
      Match: traffic-class 7
      Priority: Strict,

      Priority Level: 1
      shape (average) cir 3000000000, bc 30000000, be 30000000
      target shape rate 3000000000
```

```
Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
  0 packets
  Match: traffic-class 6
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  shape (average) cir 1000000000, bc 4000000, be 4000000
  target shape rate 1000000000
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0              85     100  1/1
    1              65     100  1/1
    2               0       0  1/1
    3               0       0  1/1
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
  0 packets
  Match: traffic-class 5
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0              85     100  1/1
    1              65     100  1/1
    2               0       0  1/1
    3               0       0  1/1
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
  0 packets
  Match: traffic-class 4
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0              85     100  1/1
    1              65     100  1/1
    2               0       0  1/1
    3               0       0  1/1
```

```
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
  0 packets
  Match: traffic-class 3
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 4

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0               85     100  1/1
    1               65     100  1/1
    2                0       0  1/1
    3                0       0  1/1
    4                0       0  1/1
    5                0       0  1/1
    6                0       0  1/1
    7                0       0  1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
  0 packets
  Match: traffic-class 2
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 1

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0               85     100  1/1
    1               65     100  1/1
    2                0       0  1/1
    3                0       0  1/1
    4                0       0  1/1
    5                0       0  1/1
    6                0       0  1/1
    7                0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
  0 packets
  Match: traffic-class 1
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0               85     100  1/1
    1               65     100  1/1
```

```
    2               0       0  1/1
    3               0       0  1/1
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1

  Class-map: class-default (match-any)
    19 packets
    Match: any
    Queueing
    queue limit 5000000 bytes
    (total drops) 0
    (bytes output) 0
    bandwidth remaining ratio 23

      Exp-weight-constant: 1 (1/2)
      Mean queue depth: 0
          Minimum Maximum  Mark
                  thresh  thresh  prob
    0               85     100  1/1
    1               65     100  1/1
    2                0       0  1/1
    3                0       0  1/1
    4                0       0  1/1
    5                0       0  1/1
    6                0       0  1/1
    7                0       0  1/1
```

Example: auto qos voip trust

This is an example of the **auto qos voip trust** command and the applied policies and class maps.

These policy maps are created and applied when running this command:`do sh ru interface FiftyGigE1/0/1`

```
Device(config-if)#do sh ru interface FiftyGigE1/0/30
nterface FiftyGigE1/0/30
 auto qos trust
 service-policy input AutoQos-4.0-Trust-Cos-Input-Policy
 service-policy type queueing output AutoQos-4.0-Output-Policy

Device(config-if)#do sh policy-map interface FiftyGigE1/0/1
FiftyGigE1/0/1
Service-policy input: AutoQos-4.0-Trust-Cos-Input-Policy
Class-map: class-default (match-any)
0 packets
Match: any
QoS Set
traffic-class cos table AutoQos-4.0-Trust-Cos-Table
police:
cir 5000000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
```

These class maps are created and applied when running this command: `do sh policy-map ty q interface FiftyGigE1/0/1 | in Auto`

```
Device(config-if)#do sh policy-map ty q interface FiftyGigE1/0/1 | in Class-map:
Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
```

```
Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
Class-map: class-default (match-any)
```

This is a sample output of the **show policy-map interface** command for a Layer 2 interface. Depending on the type of interface (Layer 3 or Layer 2), QoS policies from trust DSCP or CoS will be installed.

```
Device(config-if)#do sh policy-map interface FiftyGigE1/0/1
FiftyGigE1/0/1
Service-policy input: AutoQos-4.0-Trust-Cos-Input-Policy
Class-map: class-default (match-any)
0 packets
Match: any
QoS Set
traffic-class cos table AutoQos-4.0-Trust-Cos-Table
police:
cir 5000000000 bps, bc 1024000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
```

This is a sample output of the **show policy-map type queueing interface** command.

```
Device(config)# interface FiftyGigE 6/0/1
Device(config-if)# auto qos voip trust
Device(config-if)# end
Device(config-if)#do sh policy-map type queueing interface FiftyGigE1/0/1
FiftyGigE1/0/1

Service-policy queueing output: AutoQos-4.0-Output-Policy

queue stats for all priority classes:
Queueing
priority level 1
queue limit 32256 bytes
(total drops) 0
(bytes output) 0

Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
0 packets
Match: traffic-class 7
Priority: Strict,

Priority Level: 1
shape (average) cir 30000000000, bc 300000000, be 300000000
target shape rate 30000000000

Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
0 packets
Match: traffic-class 6
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
```

```
Minimum Maximum  Mark
thresh  thresh  prob
0               85     100  1/1
1               65     100  1/1
2                0       0  1/1
3                0       0  1/1
4                0       0  1/1
5                0       0  1/1
6                0       0  1/1
7                0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
0 packets
Match: traffic-class 5
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0               85     100  1/1
1               65     100  1/1
2                0       0  1/1
3                0       0  1/1
4                0       0  1/1
5                0       0  1/1
6                0       0  1/1
7                0       0  1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
0 packets
Match: traffic-class 4
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0               85     100  1/1
1               65     100  1/1
2                0       0  1/1
3                0       0  1/1
4                0       0  1/1
5                0       0  1/1
6                0       0  1/1
7                0       0  1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
0 packets
Match: traffic-class 3
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 4
```

```
Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
0 packets
Match: traffic-class 2
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 1

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
0 packets
Match: traffic-class 1
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: class-default (match-any)
0 packets
Match: any
Queueing
queue limit 5000000 bytes
(total drops) 0
(bytes output) 0
```

```
bandwidth remaining ratio 23

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1
```

Example: auto qos voip cisco-phone

This is an example of the **auto qos voip cisco-phone** command and the applied policies and class maps.

These policy maps are created and applied when running this command:

- AutoQos-4.0-CiscoPhone-Input-Policy
- AutoQos-4.0-Output-Policy

These class maps are created and applied when running this command: **do sh policy-map interface FiftyGigE1/0/1 | in Class-map**

```
Device(config-if)#do sh policy-map interface FiftyGigE1/0/1 | in Class-map:
        Class-map: AutoQos-4.0-Voip-Data-CiscoPhone-Class (match-any)
        Class-map: AutoQos-4.0-Voip-Signal-CiscoPhone-Class (match-any)
        Class-map: AutoQos-4.0-Default-Class (match-any)
        Class-map: class-default (match-any)
```

These class maps are created and applied when running this command: `do sh policy-map ty q interface FiftyGigE1/0/1 | in Auto`

- AutoQos-4.0-Output-Policy
- AutoQos-4.0-Output-Priority-Queue (match-any)
- AutoQos-4.0-Output-Signaling-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
- AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
- AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
- AutoQos-4.0-Output-Scavenger-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)

```
Device(config)# interface FiftyGigE1/0/1
Device(config-if)# auto qos voip cisco-phone
Device(config-if)# end

Device(config-if)#do sh policy-map interface FiftyGigE1/0/1
FiftyGigE1/0/1
Service-policy input: AutoQos-4.0-CiscoPhone-Input-Policy
Class-map: AutoQos-4.0-Voip-Data-CiscoPhone-Class (match-any)
```

```
        0 packets
        Match: cos 5
        QoS Set
        traffic-class 7
        police:
        cir 5 %
        cir 500000000 bps, bc 15625000 bytes
        conformed 0 bytes; actions:
        transmit
        exceeded 0 bytes; actions:
        set-discard-class-transmit 1
        conformed 0000 bps, exceeded 0000 bps
        Class-map: AutoQos-4.0-Voip-Signal-CiscoPhone-Class (match-any)
        0 packets
        Match: cos 3
        QoS Set
        traffic-class 6
        police:
        cir 1 %
        cir 100000000 bps, bc 3125000 bytes
        conformed 0 bytes; actions:
        transmit
        exceeded 0 bytes; actions:
        set-discard-class-transmit 1
        conformed 0000 bps, exceeded 0000 bps
        Class-map: AutoQos-4.0-Default-Class (match-any)
        0 packets
        Match: access-group name AutoQos-4.0-Acl-Default
        QoS Set
        dscp default
        Class-map: class-default (match-any)
        0 packets
        Match: any

Device(config-if)#do sh policy-map ty q interface FiftyGigE1/0/1
FiftyGigE1/0/1
Service-policy queueing output: AutoQos-4.0-Output-Policy
queue stats for all priority classes:
Queueing
priority level 1
queue limit 32256 bytes
(total drops) 0
(bytes output) 0
Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
0 packets
Match: traffic-class 7
Priority: Strict,

Priority Level: 1
shape (average) cir 3000000000, bc 30000000, be 30000000
target shape rate 3000000000
Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
0 packets
Match: traffic-class 6
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 1000000000, bc 4000000, be 4000000
target shape rate 1000000000
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
```

```
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
0 packets
Match: traffic-class 5
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
0 packets
Match: traffic-class 4
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
0 packets
Match: traffic-class 3
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 4

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
```

```
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
0 packets
Match: traffic-class 2
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 1

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
0 packets
Match: traffic-class 1
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: class-default (match-any)
0 packets
Match: any
Queueing
queue limit 5000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 23

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
```

```
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
```

Example: auto qos voip cisco-softphone

This is an example of the **auto qos voip cisco-softphone** command and the applied policies and class maps.

These policy maps are created and applied when running this command: **do sh ru interface FiftyGigE1/0/1**

```
Device(config-if)#do sh ru interface FiftyGigE1/0/1
Building configuration...
Current configuration : 215 bytes
!
interface FiftyGigE1/0/1
auto qos voip cisco-softphone
service-policy input AutoQos-4.0-CiscoSoftPhone-Input-Policy
service-policy type queueing output AutoQos-4.0-Output-Policy
```

These class maps are created and applied when running this command:`do sh policy-map int fif 1/0/1`

- AutoQos-4.0-Voip-Data-Dscp-Class (match-any)
- AutoQos-4.0-Voip-Data-Cos-Class (match-any)
- AutoQos-4.0-Voip-Signal-Dscp-Class (match-any)
- AutoQos-4.0-Voip-Signal-Cos-Class (match-any)
- AutoQos-4.0-Signaling-Class (match-any)
- AutoQos-4.0-Multimedia-Conf-Class (match-any)
- AutoQos-4.0-Transaction-Class (match-any)
- AutoQos-4.0-Bulk-Data-Class (match-any)
- AutoQos-4.0-Scavanger-Class (match-any)
- AutoQos-4.0-Default-Class (match-any)
- Class-map: class-default (match-any)

These class maps are created and applied when running this command:`do sh policy-map ty q int fif 1/0/1`

- AutoQos-4.0-Output-Policy
- AutoQos-4.0-Output-Priority-Queue (match-any)
- AutoQos-4.0-Output-Signaling-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
- AutoQos-4.0-Output-Transaction-Data-Queue (match-any)

- AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
- AutoQos-4.0-Output-Scavenger-Queue (match-any)
- AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
- Class-map: class-default (match-any)

```
Device(config)# interface FiftyGigE1/0/1
Device(config-if)# auto qos voip cisco-softphone
Device(config-if)# end

Device(config-if)#do sh policy-map int FiftyGigE1/0/1
Device(config-if)#auto qos voip cisco-softphone
Device(config-if)#do sh policy-map interface FiftyGigE1/0/1
FiftyGigE1/0/1

Service-policy input: AutoQos-4.0-CiscoSoftPhone-Input-Policy

Class-map: AutoQos-4.0-Voip-Data-Dscp-Class (match-any)
0 packets
Match:  dscp ef (46)
QoS Set
traffic-class 7
police:
cir 4 %
cir 4000000000 bps, bc 125000000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Voip-Data-Cos-Class (match-any)
0 packets
Match: cos  5
QoS Set
traffic-class 7
police:
cir 4 %
cir 4000000000 bps, bc 125000000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Voip-Signal-Dscp-Class (match-any)
0 packets
Match:  dscp cs3 (24)
QoS Set
traffic-class 6
police:
cir 1 %
cir 1000000000 bps, bc 31250000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Voip-Signal-Cos-Class (match-any)
0 packets
```

```
Match: cos  3
QoS Set
traffic-class 6
police:
cir 1 %
cir 1000000000 bps, bc 31250000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Signaling-Class (match-any)
0 packets
Match: access-group name AutoQos-4.0-Acl-Signaling
QoS Set
traffic-class 5
dscp cs3
police:
cir 1 %
cir 1000000000 bps, bc 31250000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Multimedia-Conf-Class (match-any)
0 packets
Match: access-group name AutoQos-4.0-Acl-Multimedia-Conf
QoS Set
traffic-class 4
dscp af41
police:
cir 5 %
cir 5000000000 bps, bc 156250000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
set-dscp-transmit dscp table AutoQos-Police-Markdown-DSCP-Table
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Transaction-Class (match-any)
0 packets
Match: access-group name AutoQos-4.0-Acl-Transaction-Data
QoS Set
traffic-class 3
dscp af21
police:
cir 10 %
cir 10000000000 bps, bc 312500000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
set-dscp-transmit dscp table AutoQos-Police-Markdown-DSCP-Table
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Bulk-Data-Class (match-any)
0 packets
Match: access-group name AutoQos-4.0-Acl-Bulk-Data
QoS Set
traffic-class 2
```

```
dscp af11
police:
cir 10 %
cir 10000000000 bps, bc 312500000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
set-dscp-transmit dscp table AutoQos-Police-Markdown-DSCP-Table
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Scavanger-Class (match-any)
0 packets
Match: access-group name AutoQos-4.0-Acl-Scavanger
QoS Set
traffic-class 1
dscp cs1
police:
cir 10 %
cir 10000000000 bps, bc 312500000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
conformed 0000 bps, exceeded 0000 bps

Class-map: AutoQos-4.0-Default-Class (match-any)
0 packets
Match: access-group name AutoQos-4.0-Acl-Default
QoS Set
dscp default
police:
cir 10 %
cir 10000000000 bps, bc 312500000 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
set-discard-class-transmit 1
conformed 0000 bps, exceeded 0000 bps

Class-map: class-default (match-any)
0 packets
Match: any
```

This is a sample output of the **sh policy-map type queueing interface** command.

```
Device(config-if)#do sh policy-map type queueing interface FiftyGigE1/0/1
        FiftyGigE1/0/1

        Service-policy queueing output: AutoQos-4.0-Output-Policy

        queue stats for all priority classes:
        Queueing
        priority level 1
        queue limit 32256 bytes
        (total drops) 0
        (bytes output) 0

        Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
        0 packets
        Match: traffic-class 7
        Priority: Strict,

        Priority Level: 1
```

```
shape (average) cir 30000000000, bc 300000000, be 300000000
target shape rate 30000000000

Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
0 packets
Match: traffic-class 6
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
0 packets
Match: traffic-class 5
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
0 packets
Match: traffic-class 4
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
```

```
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
0 packets
Match: traffic-class 3
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 4

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
0 packets
Match: traffic-class 2
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 1

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
0 packets
Match: traffic-class 1
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
```

```
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1

Class-map: class-default (match-any)
0 packets
Match: any
Queueing
queue limit 5000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 23

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum  Mark
thresh  thresh  prob
0              85     100  1/1
1              65     100  1/1
2               0       0  1/1
3               0       0  1/1
4               0       0  1/1
5               0       0  1/1
6               0       0  1/1
7               0       0  1/1
```

Example: auto qos classify police

This is an example of the **auto qos classify police** command and the applied policies and class maps.

These policy maps are created and applied when running this command: **do sh ru int FiftyGigE1/0/30**

```
Device(config-if)#do sh ru int FiftyGigE1/0/30
Building configuration...

Current configuration : 186 bytes
!
interface FiftyGigE1/0/30
 auto qos classify police
 service-policy input AutoQos-4.0-Classify-Police-Input-Policy
 service-policy type queueing output AutoQos-4.0-Output-Policy
```

```
Device(config-if)#do sh policy-map int FiftyGigE1/0/30  | in Class
  Service-policy input: AutoQos-4.0-Classify-Police-Input-Policy
    Class-map: AutoQos-4.0-Signaling-Class (match-any)
    Class-map: AutoQos-4.0-Multimedia-Conf-Class (match-any)
    Class-map: AutoQos-4.0-Transaction-Class (match-any)
    Class-map: AutoQos-4.0-Bulk-Data-Class (match-any)
    Class-map: AutoQos-4.0-Scavanger-Class (match-any)
    Class-map: AutoQos-4.0-Default-Class (match-any)
    Class-map: class-default (match-any)
```

These class maps are created and applied when running this command:`do sh policy-map ty q int fif 1/0/30`

```
do sh policy-map ty queue int FiftyGigE1/0/30  | in Class
    Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
    Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
    Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
```

```
    Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
    Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
    Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
    Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
    Class-map: class-default (match-any)


do sh policy-map   int FiftyGigE1/0/30
 FiftyGigE1/0/30

  Service-policy input: AutoQos-4.0-Classify-Police-Input-Policy

    Class-map: AutoQos-4.0-Signaling-Class (match-any)
      0 packets
      Match: access-group name AutoQos-4.0-Acl-Signaling
      QoS Set
        dscp cs3
        traffic-class 6
      police:
          cir 10 %
          cir 1000000000 bps, bc 31250000 bytes
        conformed 0 bytes; actions:
          transmit
        exceeded 0 bytes; actions:
          set-discard-class-transmit 1
        conformed 0000 bps, exceeded 0000 bps

    Class-map: AutoQos-4.0-Multimedia-Conf-Class (match-any)
      0 packets
      Match: access-group name AutoQos-4.0-Acl-Multimedia-Conf
      QoS Set
        dscp af41
        traffic-class 5
      police:
          cir 10 %
          cir 1000000000 bps, bc 31250000 bytes
        conformed 0 bytes; actions:
          transmit
        exceeded 0 bytes; actions:
          set-dscp-transmit dscp table AutoQos-Police-Markdown-DSCP-Table
          set-discard-class-transmit 1
        conformed 0000 bps, exceeded 0000 bps

    Class-map: AutoQos-4.0-Transaction-Class (match-any)
      0 packets
      Match: access-group name AutoQos-4.0-Acl-Transaction-Data
      QoS Set
        dscp af21
        traffic-class 4
      police:
          cir 10 %
          cir 1000000000 bps, bc 31250000 bytes
        conformed 0 bytes; actions:
          transmit
        exceeded 0 bytes; actions:
          set-dscp-transmit dscp table AutoQos-Police-Markdown-DSCP-Table
          set-discard-class-transmit 1
        conformed 0000 bps, exceeded 0000 bps

    Class-map: AutoQos-4.0-Bulk-Data-Class (match-any)
      0 packets
      Match: access-group name AutoQos-4.0-Acl-Bulk-Data
      QoS Set
        dscp af11
        traffic-class 3
```

```
    police:
        cir 10 %
        cir 1000000000 bps, bc 31250000 bytes
      conformed 0 bytes; actions:
        transmit
      exceeded 0 bytes; actions:
        set-dscp-transmit dscp table AutoQos-Police-Markdown-DSCP-Table
        set-discard-class-transmit 1
      conformed 0000 bps, exceeded 0000 bps

  Class-map: AutoQos-4.0-Scavanger-Class (match-any)
    0 packets
    Match: access-group name AutoQos-4.0-Acl-Scavanger
    QoS Set
      dscp cs1
      traffic-class 2
    police:
        cir 10 %
        cir 1000000000 bps, bc 31250000 bytes
      conformed 0 bytes; actions:
        transmit
      exceeded 0 bytes; actions:
        set-discard-class-transmit 1
      conformed 0000 bps, exceeded 0000 bps

  Class-map: AutoQos-4.0-Default-Class (match-any)
    110 packets
    Match: access-group name AutoQos-4.0-Acl-Default
    QoS Set
      dscp default
    police:
        cir 10 %
        cir 1000000000 bps, bc 31250000 bytes
      conformed 13838 bytes; actions:
        transmit
      exceeded 0 bytes; actions:
        set-discard-class-transmit 1
      conformed 0000 bps, exceeded 0000 bps

  Class-map: class-default (match-any)
    4227 packets
    Match: any
```

This is a sample output of the **sh policy-map type queueing interface** command.

```
do sh policy-map ty q  int FiftyGigE1/0/30
 FiftyGigE1/0/30

  Service-policy queueing output: AutoQos-4.0-Output-Policy

    queue stats for all priority classes:
      Queueing
      priority level 1
      queue limit 32256 bytes
      (total drops) 0
      (bytes output) 11732

    Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
      70 packets
      Match: traffic-class 7
      Priority: Strict,

      Priority Level: 1
      shape (average) cir 3000000000, bc 30000000, be 30000000
      target shape rate 3000000000
```

```
Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
  0 packets
  Match: traffic-class 6
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  shape (average) cir 1000000000, bc 4000000, be 4000000
  target shape rate 1000000000
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0              85     100  1/1
    1              65     100  1/1
    2               0       0  1/1
    3               0       0  1/1
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
  0 packets
  Match: traffic-class 5
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0              85     100  1/1
    1              65     100  1/1
    2               0       0  1/1
    3               0       0  1/1
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
  0 packets
  Match: traffic-class 4
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0              85     100  1/1
    1              65     100  1/1
    2               0       0  1/1
    3               0       0  1/1
```

```
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
  0 packets
  Match: traffic-class 3
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 4

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0               85     100  1/1
    1               65     100  1/1
    2                0       0  1/1
    3                0       0  1/1
    4                0       0  1/1
    5                0       0  1/1
    6                0       0  1/1
    7                0       0  1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
  0 packets
  Match: traffic-class 2
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 1

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0               85     100  1/1
    1               65     100  1/1
    2                0       0  1/1
    3                0       0  1/1
    4                0       0  1/1
    5                0       0  1/1
    6                0       0  1/1
    7                0       0  1/1

Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
  0 packets
  Match: traffic-class 1
  Queueing
  queue limit 300000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 9

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0               85     100  1/1
    1               65     100  1/1
```

```
  2               0       0  1/1
  3               0       0  1/1
  4               0       0  1/1
  5               0       0  1/1
  6               0       0  1/1
  7               0       0  1/1

Class-map: class-default (match-any)
  68 packets
  Match: any
  Queueing
  queue limit 5000000 bytes
  (total drops) 0
  (bytes output) 0
  bandwidth remaining ratio 23

    Exp-weight-constant: 1 (1/2)
    Mean queue depth: 0
        Minimum Maximum  Mark
                thresh  thresh  prob
    0              85     100  1/1
    1              65     100  1/1
    2               0       0  1/1
    3               0       0  1/1
    4               0       0  1/1
    5               0       0  1/1
    6               0       0  1/1
    7               0       0  1/1
```

Example: auto qos global compact

This is an example of the **auto qos global compact** command.

```
Device# configure terminal
Device(config)# auto qos global compact
Device(config)# interface FiftyGigE1/0/30
Device(config-if)# auto qos voip trust

Device# show auto qos

FiftyGigE1/0/30
auto qos voip trust

Device# show running-config interface FiftyGigE1/0/30
interface FiftyGigE1/0/30
auto qos trust
service-policy input AutoQos-4.0-Trust-Cos-Input-Policy
service-policy type queueing output AutoQos-4.0-Output-Policy
```

This is a sample output of the `do sh policy-map interface FiftyGigE1/0/30` command

```
Device(config-if)#do sh policy-map interface FiftyGigE1/0/30
        FiftyGigE1/0/30
        Service-policy input: AutoQos-4.0-Trust-Cos-Input-Policy
        Class-map: class-default (match-any)
        0 packets
        Match: any
        QoS Set
        traffic-class cos table AutoQos-4.0-Trust-Cos-Table
        police:
        cir 5000000000 bps, bc 1024000 bytes
```

```
        conformed 0 bytes; actions:
        transmit
        exceeded 0 bytes; actions:
        drop
        conformed 0000 bps, exceeded 0000 bps
```

This is a sample output of the `do sh policy-map ty q interface FiftyGigE1/0/30` command

```
Device(config-if)#do sh policy-map ty q interface FiftyGigE1/0/30
        FiftyGigE1/0/30
        Service-policy queueing output: AutoQos-4.0-Output-Policy
        queue stats for all priority classes:
        Queueing
        priority level 1
        queue limit 32256 bytes
        (total drops) 0
        (bytes output) 0
        Class-map: AutoQos-4.0-Output-Priority-Queue (match-any)
        0 packets
        Match: traffic-class 7
        Priority: Strict,

        Priority Level: 1
        shape (average) cir 3000000000, bc 30000000, be 30000000
        target shape rate 3000000000
        Class-map: AutoQos-4.0-Output-Signaling-Queue (match-any)
        0 packets
        Match: traffic-class 6
        Queueing
        queue limit 300000 bytes
        (total drops) 0
        (bytes output) 0
        shape (average) cir 1000000000, bc 4000000, be 4000000
        target shape rate 1000000000
        bandwidth remaining ratio 9

        Exp-weight-constant: 1 (1/2)
        Mean queue depth: 0
        Minimum Maximum Mark
        thresh thresh prob
        0 85 100 1/1
        1 65 100 1/1
        2 0 0 1/1
        3 0 0 1/1
        4 0 0 1/1
        5 0 0 1/1
        6 0 0 1/1
        7 0 0 1/1

        Class-map: AutoQos-4.0-Output-Multimedia-Conf-Queue (match-any)
        0 packets
        Match: traffic-class 5
        Queueing
        queue limit 300000 bytes
        (total drops) 0
        (bytes output) 0
        bandwidth remaining ratio 9

        Exp-weight-constant: 1 (1/2)
        Mean queue depth: 0
        Minimum Maximum Mark
        thresh thresh prob
        0 85 100 1/1
        1 65 100 1/1
        2 0 0 1/1
```

```
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Transaction-Data-Queue (match-any)
0 packets
Match: traffic-class 4
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Bulk-Data-Queue (match-any)
0 packets
Match: traffic-class 3
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 4

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Scavenger-Queue (match-any)
0 packets
Match: traffic-class 2
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 1

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
```

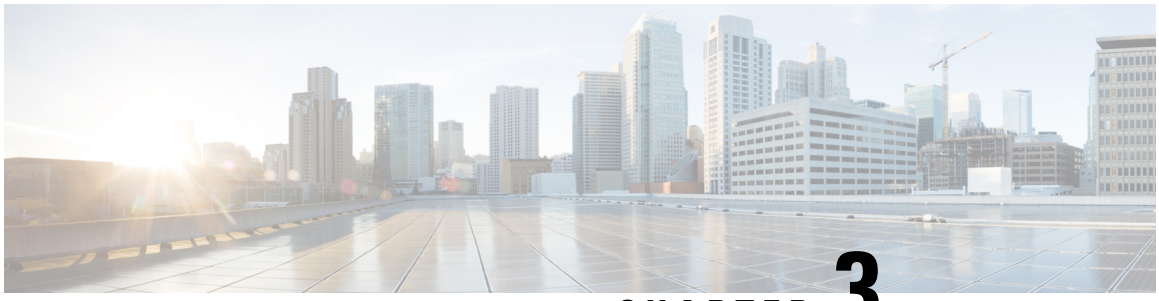
```
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: AutoQos-4.0-Output-Multimedia-Stream-Queue (match-any)
0 packets
Match: traffic-class 1
Queueing
queue limit 300000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 9

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1

Class-map: class-default (match-any)
0 packets
Match: any
Queueing
queue limit 5000000 bytes
(total drops) 0
(bytes output) 0
bandwidth remaining ratio 23

Exp-weight-constant: 1 (1/2)
Mean queue depth: 0
Minimum Maximum Mark
thresh thresh prob
0 85 100 1/1
1 65 100 1/1
2 0 0 1/1
3 0 0 1/1
4 0 0 1/1
5 0 0 1/1
6 0 0 1/1
7 0 0 1/1
```



CHAPTER 3

HQoS

This document provides detailed understanding about HQoS and its configuration.

- Feature History for HQoS, on page 139
- Information About HQoS, on page 139
- Prerequisites for HQoS, on page 140
- Restrictions for HQoS, on page 141
- Limitation for HQoS, on page 141
- How to Configure HQoS, on page 141
- Configuration Samples for HQoS, on page 141

Feature History for HQoS

This table provides release and related information for features explained in this module.

These features are available on all releases subsequent to the one they were introduced in, unless noted otherwise.

Table 11: Feature History for HQoS

Release	Feature	Feature Information
Cisco IOS XE 26.2.1ea	HQoS Functionality: Hierarchical Quality of Service (HQoS) is a technology used to guarantee the bandwidth of multiple services of many users in the Differentiated Service (DiffServ) model through a queue scheduling mechanism.	Cisco C9550 Series Smart Switches

Information About HQoS

Information About HQoS

Hierarchical QoS (HQoS) on Cisco switches is a traffic management method that provides granular control by applying multiple levels of Quality of Service (QoS) policies. It allows for a more complex prioritization scheme than a single-level policy by using a hierarchy of policies, such as parent and child policies, to classify,

police, shape, and queue traffic based on different requirements. This allows you to manage bandwidth and prioritize traffic more effectively across different services and levels of your network.

Hierarchical policy structure

HQoS uses a multi-level structure, most commonly a two-level hierarchy of a parent and child policy.

Parent level

This is the top-level policy, often applied to a physical port, which manages overall bandwidth allocation and can have a child policy attached to it.

Child level

This is a nested policy that applies to specific traffic classes, which are identified at the parent level. The child policy can then perform detailed actions like queueing or marking for that specific traffic.

Granular traffic management

By nesting policies, HQoS allows for more sophisticated control. For example, a parent policy might shape the total bandwidth for a port, while a child policy within it can prioritize voice traffic over video traffic within that shaped bandwidth.

Support for multiple actions

HQoS supports performing different actions at different levels of the hierarchy. For example, you can have a parent policy that shapes traffic and a child policy that performs queuing.

Prerequisites for HQoS

Before configuring standard Hierarchical Quality of Service (QoS), you must have a thorough understanding of these items:

- Standard QoS concepts.
- Classic Cisco IOS QoS.
- Modular QoS CLI (MQC).
- Understanding of Virtual Output Queuing (VOQ) architecture.
- The types of applications used and the traffic patterns on your network.
- Traffic characteristics and needs of your network. For example, is the traffic on your network bursty? Do you need to reserve bandwidth for voice and video streams?
- Bandwidth requirements and speed of the network.
- Location of congestion points in the network.

Restrictions for HQoS

In a HQoS policy with parent shaping and child policy having priority level queuing and priority level policing, the statistics for policing are not updated. Only HQoS shaper statistics are updated. To view the HQoS shaper statistics, use the `show policy-map type queuing interface <interface name>` command in global configuration mode.

Limitation for HQoS

- For hierarchical QoS (HQoS) police applied in the ingress direction, child-level policing is supported. Policing of traffic is based on the police rate value defined in parent-level policing.
- Bandwidth remaining ratio is not supported for parent policy.

How to Configure HQoS

For information about child policy, refer to How to Configure QoS.

Configuration Samples for HQoS

The following sections provide configuration examples for HQoS.

The configuration for Ingress HQoS is as follows:

```
policy-map child-policy
class cs1
set dscp cs2
class cs2
set dscp cs3
```

Note Child policy supports only remarking.

```
policy-map parent
class class-default
police cir 2g
service-policy child-policy
```

Note This policy applies to ingress HQoS.

Ingress HQoS output

The configuration output for ingress HQoS is as follows:

```
VTEP1(config-if)#do sh policy-map
int FortyGigabitEthernet6/0/2 in
FortyGigabitEthernet6/0/2
```

Service-policy input: parent

The configuration for service-policy input for a parent is as follows:

```
Class-map: class-default (match-any)
24982466 packets
Match: any
police:
cir 2000000000 bps, bc 62500000 bytes
conformed 19893033000 bytes; actions:
transmit
exceeded 11334205500 bytes; actions:
drop
conformed 418702000 bps, exceeded 238496000 bps
```

Service-policy: remark

The configuration for service-policy remark is as follows:

```
Device(config-if)#do sh policy-map int FiftyGigE1/0/30
FiftyGigE1/0/30
Service-policy input: parent
Class-map: class-default (match-any)
0 packets
Match: any
police:
cir 200000000 bps, bc 10240 bytes
conformed 0 bytes; actions:
transmit
exceeded 0 bytes; actions:
drop
conformed 0000 bps, exceeded 0000 bps
Service-policy : remark
Class-map: cs1 (match-all)
0 packets
Match: dscp cs1 (8)
QoS Set
dscp cs3
Class-map: cs2 (match-all)
0 packets
Match: dscp cs2 (16)
QoS Set
dscp cs2
Class-map: cs5 (match-all)
0 packets
Match: dscp cs5 (40)
QoS Set
dscp cs3
Class-map: cs4 (match-all)
0 packets
Match: dscp cs4 (32)
QoS Set
dscp cs7
Class-map: cs3 (match-all)
0 packets
Match: dscp cs3 (24)
QoS Set
dscp cs3
Class-map: cs7 (match-all)
0 packets
```

```
Match: dscp cs7 (56)
QoS Set
dscp 1
Class-map: cs6 (match-all)
0 packets
Match: dscp cs6 (48)
QoS Set
dscp 3
Class-map: class-default (match-any)
0 packets
Match: any
Device(config-if)#
```

Child policy configuration

The configuration for queueing HQoS is as follows:

```
policy-map type queueing child-llq
class tc7
priority level 1
shape average 20g
class tc6
shape average percent 1
class tc5
shape average 15g
class tc4
shape average 5g
class tc3
shape average 15g
class tc2
shape average 15g
class tc1
shape average 4g
class class-default
shape average 10g
```

Parent policy configuration

The configuration for parent policy is as follows:

```
policy-map type queueing parent-llq
class class-default
shape average 80g
service-policy child-llq
```

Note The parent shape rate value must be greater than 1.35g. If it is less than 1.35g, the configuration will fail with an error message.

For information about child policy, refer to How to Configure QoS.

Note For hierarchical QoS (HQoS) policy applied in the ingress direction, child-level policing is supported. Policing of traffic is based on the police rate value defined in parent-level policing.

HQoS configuration and interface statistics

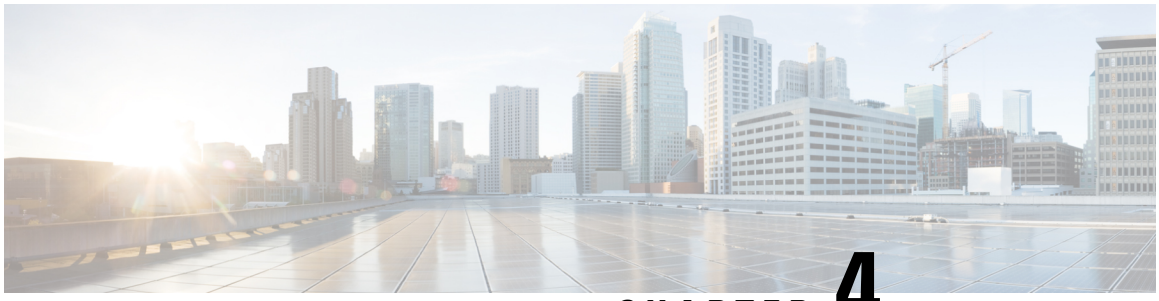
To view the HQoS configuration and statistics on an interface, you can use the following CLI command:

```
show policy-map type queue interfae FiftyGigE1/0/3
```

The configuration output for HQoS configuration and statistics is as follows:

```
Device(config-if)#do sh policy-map ty q int FiftyGigE1/0/30
FiftyGigE1/0/30
Service-policy queueing output: hqos
Class-map: class-default (match-any)
5 packets
Match: any
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 619
shape (average) cir 3000000000, bc 12000000, be 12000000
target shape rate 3000000000
Service-policy queueing : cosllq
queue stats for all priority classes:
Queueing
priority level 1
queue limit 32256 bytes
(total drops) 0
(bytes output) 619
queue stats for all priority classes:
Queueing
priority level 2
queue limit 575999 bytes
(total drops) 0
(bytes output) 0
Class-map: tc7 (match-all)
2 packets
Match: traffic-class 7
shape (average) cir 100000000, bc 400000, be 400000
target shape rate 100000000
Priority: Strict,
Priority Level: 1
Class-map: tc6 (match-all)
0 packets
Match: traffic-class 6
shape (average) cir 1000000000, bc 4000000, be 4000000
target shape rate 1000000000
Priority: Strict,
Priority Level: 2
Class-map: tc5 (match-all)
0 packets
Match: traffic-class 5
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 30000000, bc 300000, be 300000
target shape rate 30000000
Class-map: tc4 (match-all)
0 packets
Match: traffic-class 4
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 1000000000, bc 4000000, be 4000000
target shape rate 1000000000
```

```
Class-map: tc3 (match-all)
0 packets
Match: traffic-class 3
Queueing
queue limit 1000000 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 1000000000, bc 4000000, be 4000000
target shape rate 1000000000
Class-map: tc2 (match-all)
0 packets
Match: traffic-class 2
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 1000000000, bc 4000000, be 4000000
target shape rate 1000000000
Class-map: tc1 (match-all)
0 packets
Match: traffic-class 1
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 1500000000, bc 6000000, be 6000000
target shape rate 1500000000
Class-map: class-default (match-any)
3 packets
Match: any
Queueing
queue limit 98304 bytes
(total drops) 0
(bytes output) 0
shape (average) cir 100000000000, bc 400000000, be 400000000
target shape rate 100000000000
```



CHAPTER 4

Layer 3 subinterface queuing

This document provides detailed understanding about Layer 3 subinterface queuing and its configuration.

- Feature history for Layer 3 Subinterface Queuing, on page 147
- Default queuing mode, on page 147
- Subinterface propagation queuing mode, on page 148
- Hierarchical QoS, on page 149
- Limitations for Layer 3 subinterface queuing, on page 150
- Enabling subinterface queuing policy, on page 151
- Enabling subinterface priority propagation mode, on page 154
- Configuring hierarchical QoS policy on a subinterface, on page 155
- Configuration examples, on page 158
- Monitor Layer 3 subinterface queuing, on page 160

Feature history for Layer 3 Subinterface Queuing

This table provides release and platform support information for the features explained in this module.

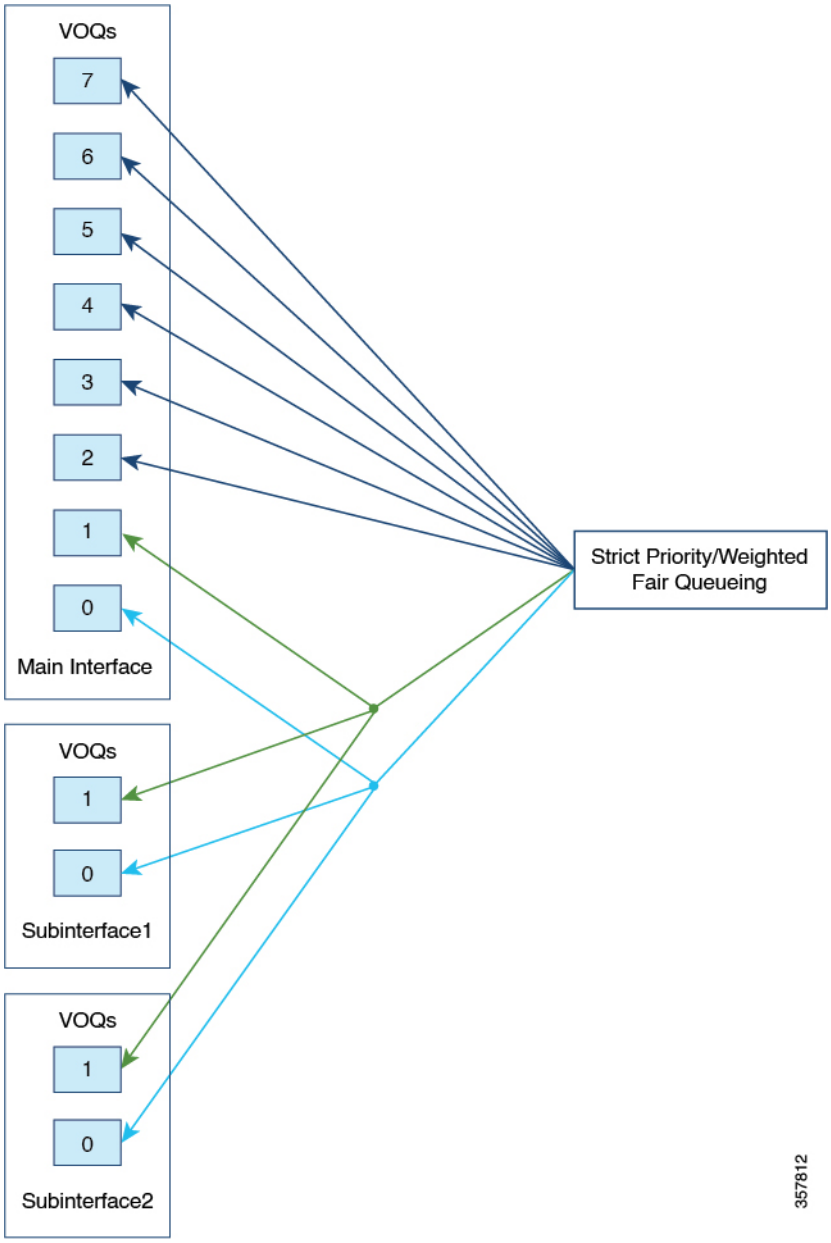
These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Release	Feature name and description	Supported platform
Cisco IOS XE 17.18.1	Layer 3 Subinterface Queuing: Queuing support on Layer 3 subinterfaces.	Cisco C9350 Series Smart Switches

Default queuing mode

In default queuing mode, the main interface queue configuration has absolute priority control over the subinterface queues. The main interface has eight queues (7 to 0 in order) by default, which can be configured using [pP(8-p)Q, where p=1 to 7, P=priority, and Q=queue] combination. A subinterface has two queues (Q1 and Q0) that map to or share the priority level with the queues (Q1 and Q0) of the main interface. Therefore, these subinterface queues are lower in priority when compared to the main interface queues, based on the policy applied on the main interface.

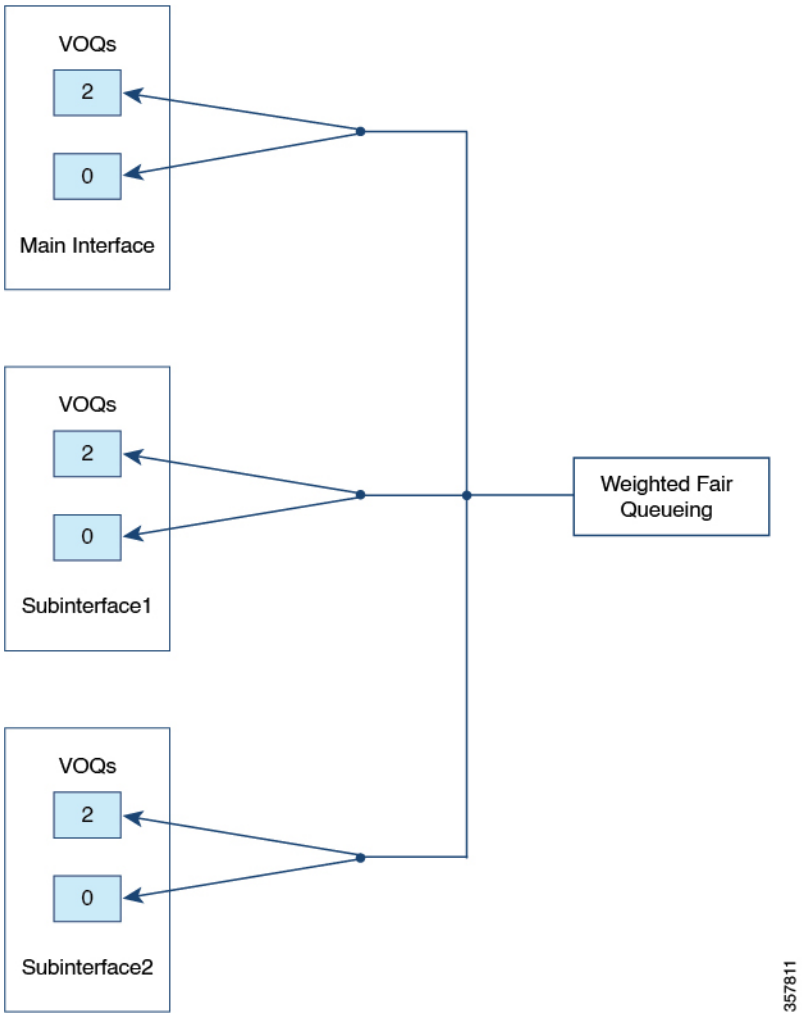
Figure 2: Default queuing mode



Subinterface propagation queuing mode

In this mode, the main interface queues are treated and configured just as any other subinterface queue, and all the behavior and restrictions of the subinterface queues are applicable to the main interface queue as well. A queue of the same priority level across the main interface and subinterfaces contend for bandwidth or priority at the same level. Further, in this mode a user can configure bandwidth distribution ratio between the main interface and subinterfaces. A user cannot change the mode of the main interface when queuing policy is applied on the main interface.

Figure 3: Subinterface propagation queuing mode



Hierarchical QoS

Hierarchical QoS (HQoS) manages network traffic by applying QoS policies at multiple levels to enhances bandwidth utilization, traffic prioritization, and overall network performance.

HQoS allows you to perform:

- Multiple levels of QoS management: HQoS manages QoS at three levels—physical interface, logical interface, and class level—enabling integrated queueing and shaping across these layers.
- Hierarchical classification: Traffic classification can be based on other classes, allowing combination of multiple classification criteria into a single class map.
- Hierarchical policing and shaping: Policing and shaping can be applied at multiple levels in the hierarchy, such as shaping at the parent level and policing at child levels.

- Integrated queueing hierarchy: HQoS supports forming an integrated queueing hierarchy on physical interfaces, which is useful in complex network architectures like Ethernet DSL access networks.
- Benefits: Fast deployment of QoS in large-scale networks; consistent behavior across Cisco platforms; support for multiple scheduling levels; ability to apply fair queuing and drop policies per class.

HQoS configuration options

HQoS configuration options include:

- Port shaper: Port shaper is used to limit or shape the overall transmission rate for a given port (main or subinterface). Port shaper is supported through a HQoS queuing policy on the main interface and subinterfaces with the shaper configured on the parent policy.
- Main interface: Port shaper applied on the main interface is applicable to all subinterfaces.
- Subinterface: Port shaper is applied only to the subinterface on which the port shaper policy is applied.
- Port bandwidth distribution ratio among subinterfaces: Port bandwidth remaining ratio is used to govern or configure the bandwidth distribution ratio between subinterfaces. Port bandwidth remaining ratio is supported with the bandwidth remaining ratio command that is configured on the parent policy applied on the subinterface.

  Port bandwidth remaining ratio is also supported between the main interface and subinterface when subinterface priority propagation mode is enabled. A maximum of seven ratio values are supported for bandwidth distribution across subinterfaces.



Note

- Hierarchical queueing features are supported only on egress; service policies with queueing cannot coexist simultaneously on child and parent interfaces.
- Hierarchical shaping supports only the port shaper. The parent configuration includes only the class default, and shaping is the sole default action.
- The value of a port shaper set on the main interface is the maximum allowed value on subinterfaces even if a higher port shaper value is defined on the subinterfaces.

Limitations for Layer 3 subinterface queuing

- Subinterface queuing is supported only on Layer 3 subinterfaces.
- Queuing policy is not supported on port channels or subinterfaces of port channels.
- Subinterface queuing supports a maximum of two queues—priority traffic class (class 7) and nonpriority traffic class (class default).
- Subinterface queuing is not supported for multicast traffic.
- Queued traffic on subinterfaces cannot be re-marked on the main interface using the re-mark policy.

- Hierarchical queuing is supported on the subinterface with this restriction—the parent policy can contain only class-default, which can have shape or bandwidth remaining ratio or both shape and bandwidth remaining ratio.
- Shaping is only supported in:
  - Priority traffic class for child or non-HQoS queuing policy.
  - Class-default for parent policy in HQoS policy.
- Bandwidth remaining ratio is not supported for queue policies with priority level defined.
- Bandwidth remaining ratio is only supported in the parent class of the HQoS queuing policy on subinterfaces.
- Traffic on all the subinterfaces without queuing policy applied, flows through the main interface queues, while traffic for the subinterfaces with queuing policy applied, flows through their respective queues.
- For direct, connected interfaces with defined IP addresses, traffic does not flow through subinterface queues but through the main interface queue. For indirect connected interfaces, traffic flows through subinterface queues.
- The **no switchport** command should be run to configure layer 3 subinterfaces.
- If a policy is applied on any of the subinterfaces, you cannot apply or remove a policy on the main interface without removing the policy on the subinterface. You can only modify the policy on the main interface.
- Queuing policy maps can be applied to a maximum of 400 Layer 3 subinterfaces.
- When HQoS is enabled on a subinterface, and the parent is configured with shape rate value (in percent), then the parent shape rate value is calculated using the main interface physical bandwidth as the reference bandwidth.
- When queuing policy is enabled on a subinterface, the subinterface queuing packet drop statistics are not reflected in the total packet drop statistics at the main interface level.

Enabling subinterface queuing policy

This procedure describes how to enable subinterface queuing policy.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password, if prompted.
Step 2	**configure terminal** **Example:**	Enters global configuration mode.

	Command or Action	Purpose
	`Device# configure terminal`	
Step 3	**class-map** {**match-any** | **match-all**} {*class name*} **Example:** `Device(config)# class-map match-all traffic-class7`	Enters class map configuration mode, and creates a class map to be used for matching packets to the class whose name you specify. • **match-any**: At least one of the match criteria must be met for traffic entering the traffic class to be classified as being a part of the traffic class. • **match-all**: All of the match criteria must be met for traffic entering the traffic class to be classified as part of the traffic class. **Note** **match-all** is the default. If **match-any** or **match-all** is not explicitly defined, **match-all** is chosen by default.
Step 4	**match traffic-class** *traffic class value* **Example:** `Device(config-cmap)# match traffic-class 7`	Matches QoS traffic class value (from 1 to 7). **Note** Only traffic class 7 is supported for queueing on subinterfaces. If it is not a subinterface queue, you can perform steps 3 and 4 to create class maps for all the traffic classes (7 to 1), and attach these to policy-map queues.
Step 5	**exit** **Example:** `Device(config-cmap)# exit`	Exits class map configuration mode and enters global configuration mode.
Step 6	**policy-map type queueing** *policy name* **Example:** `Device(config)# policy-map type queueing subif_q_policy`	Specifies the name of the main interface queueing profile policy and enters policy map configuration mode.
Step 7	**class** *class-name* **Example:** `Device(config-pmap)# class traffic-class7`	Specifies the name of the class to be associated with the policy and enters policy class map configuration mode. Command options for policy class map configuration mode include the following: • *word*: Class map name. • **class-default**: System default class matching otherwise unclassified packets, if any.

	Command or Action	Purpose
Step 8	**shape average** {*Kb/s* | **percent**} **Example:** `Device(config-pmap-c)# shape average percent 10`	Configures the traffic shaping average. The parameters include: • *Kb/s*: Use this argument to configure a specific value. The range is 1.2 Mbps to 400 Gbps. • **percent**: Allocates a maximum bandwidth to a particular class. The queue can oversubscribe bandwidth in case other queues do not utilize the entire port bandwidth. The total sum cannot exceed 100 percent, and in case it is less than 100 percent, the rest of the bandwidth is divided along all nonpriority queues based on the bandwidth remaining ratio.
Step 9	**priority level** *level* **Example:** `Device(config-pmap-c)# priority level 1`	Configures a multilevel priority queue.
Step 10	**exit** **Example:** `Device(config-pmap-c)# exit`	Exits policy class map configuration mode and enters class map configuration mode.
Step 11	**exit** **Example:** `Device(config-pmap)# exit`	Exits policy map configuration mode and enters global configuration mode.
Step 12	**interface** *interface-id* **Example:** `Device(config)# interface HundredGigE1/0/9`	Identifies the main interface and enters interface configuration mode.
Step 13	**no switchport** **Example:** `Device(config-if)# no switchport`	Switches the interface that is in Layer 2 mode into Layer 3 mode for Layer 3 configuration.
Step 14	**exit** **Example:** `Device(config-if)# exit`	Exits interface configuration mode and enters global configuration mode.

	Command or Action	Purpose
Step 15	**interface** *interface-id.subinterface-id* **Example:** `Device(config)# interface HundredGigE1/0/9.1`	Identifies the subinterface and enters subinterface configuration mode.
Step 16	**encapsulation dot1Q** *vlan-id* **Example:** `Device(config-subif)# encapsulation dot1Q 11`	Enables IEEE 802.1Q encapsulation of traffic on the subinterface.
Step 17	**service-policy type queueing output** *policy name* **Example:** `Device(config-subif)# service-policy type queueing output subif_q_policy`	Attaches two queue policy maps to the subinterface.
Step 18	**end** **Example:** `Device(config-subif)# end`	Returns to privileged EXEC mode.

Enabling subinterface priority propagation mode

This procedure describes how to enable subinterface priority propagation mode on the main interface.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password, if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**interface** *interface-id* **Example:**	Identifies the main interface and enters interface configuration mode.

	Command or Action	Purpose
	`Device(config)# interface HundredGigE1/0/23`	
Step 4	**no switchport** **Example:** `Device(config-if)# no switchport`	Switches the main interface that is in Layer 2 mode to Layer 3 mode for Layer 3 configuration.
Step 5	**queuing mode sub-interface priority-propagation** **Example:** `Device(config-if)# queuing mode sub-interface priority-propagation`	Enables subinterface priority propagation mode. **Note** This mode can be enabled only when no policy is applied on the main interface.
Step 6	**end** **Example:** `Device(config-subif)# end`	Exits subinterface configuration mode and returns to privileged EXEC mode.

Configuring hierarchical QoS policy on a subinterface

This procedure describes how to configure HQoS policy on a subinterface.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password, if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**class-map match-all** {*classname*} **Example:** `Device(config)# class-map match-all traffic-class7`	Configures the class map to match all the criteria for traffic entering the traffic class, and enters class map configuration mode.

	Command or Action	Purpose
Step 4	**match traffic-class** *class value* **Example:** `Device(config-cmap)# match traffic-class 7`	Matches QoS traffic class value.
Step 5	**exit** **Example:** `Device(config-cmap)# exit`	Exits class map configuration mode and enters global configuration mode.
Step 6	**policy-map type queueing** *child policy name* **Example:** `Device(config)# policy-map type queueing child`	Specifies the child queueing profile policy and enters policy map configuration mode.
Step 7	**class** *class-name* **Example:** `Device(config-pmap)# class traffic-class7`	Specifies the name of the class to be associated with the policy and enters policy class map configuration mode. Command options for policy class map configuration mode include the following: • *word*: Class map name. • **class-default**: System default class matching any otherwise unclassified packets.
Step 8	**shape average** {*Kb/s* | **percent**} **Example:** `Device(config-pmap-c)# shape average 1000000`	Configures the traffic shaping average. The parameters include: • *Kb/s*: Use this command to configure a specific value. The range is 1.2 Mbps to 400 Gbps. • **percent**: Allocates a maximum bandwidth to a particular class.
Step 9	**priority level** *level* **Example:** `Device(config-pmap-c)# priority level 1`	Specifies the priority level of the queue.
Step 10	**exit** **Example:** `Device(config-pmap-c)# exit`	Exits policy class map configuration mode and enters class map configuration mode.

	Command or Action	Purpose
Step 11	**exit** **Example:** `Device(config-pmap)# exit`	Exits policy map configuration mode and enters global configuration mode.
Step 12	**policy-map type queueing** *parent policy name* **Example:** `Device(config)# policy-map type queueing parent`	Specifies the parent queueing profile policy and enters policy map configuration mode.
Step 13	**class class-default** **Example:** `Device(config-pmap)# class class-default`	Specifies the class default to be associated with the parent policy and enters policy class map configuration mode.
Step 14	**shape average** {*port shaper value*} {*Kb/s* | **percent**} **Example:** `Device(config-pmap-c)# shape average 2000000000`	Configures the traffic shaping average. **Note** The shaping average must be greater than 1.35G.
Step 15	**service-policy** *child policy-map name* **Example:** `Device(config-pmap-c)# service-policy child`	Configures the QoS service policy of a child.
Step 16	**exit** **Example:** `Device(config-pmap-c)# exit`	Exits policy class map configuration mode and enters class map configuration mode.
Step 17	**exit** **Example:** `Device(config-pmap)# exit`	Exits policy map configuration mode and enters global configuration mode.
Step 18	**interface** *interface-id* **Example:** `Device(config)# interface HundredGigE1/0/5`	Identifies the interface and enters interface configuration mode.
Step 19	**no switchport** **Example:**	Switches the interface that is in Layer 2 mode into Layer 3 mode for Layer 3 configuration.

	Command or Action	Purpose
	`Device(config-if)# no switchport`	
Step 20	**exit** **Example:** `Device(config-pmap)# exit`	Exits interface configuration mode and enters global configuration mode.
Step 21	**interface** *interface-id.subinterface-id* **Example:** `Device(config)# interface HundredGigE1/0/5.1`	Identifies the subinterface and enters subinterface configuration mode.
Step 22	**encapsulation dot1Q** *vlan-id* **Example:** `Device(config-if)# encapsulation dot1Q 11`	Enables IEEE 802.1Q encapsulation of traffic on a subinterface.
Step 23	**service-policy type queueing output** *parent policy name* **Example:** `Device(config-if)# service-policy type queueing output parent`	Attaches the parent queue policy map to the subinterface.
Step 24	**end** **Example:** `Device(config-if)# end`	Returns to privileged EXEC mode.

Configuration examples

This is an example of how to enable the subinterface queuing policy:

```
Device# configure terminal
Device(config)# class-map match-all traffic-class7
Device(config-cmap)# match traffic-class 7
Device(config)# policy-map type queueing llq
Device(config-pmap)# class traffic-class7
Device(config-pmap-c)# shape average percent 10
Device(config-pmap-c)# priority level 1
Device(config-pmap-c)# exit
Device(config-pmap)# exit
Device(config)# interface HundredGigE1/0/9
Device(config-if)# no switchport
Device(config-if)# exit
Device(config)# interface HundredGigE1/0/9.1
Device(config-subif)# encapsulation dot1Q 11
```

```
Device(config-subif)# service-policy type queueing output subif_q_policy
Device(config-subif)# end
```

This is an example of how to enable the subinterface priority propagation mode:

```
Device# configure terminal
Device(config)# interface HundredGigE1/0/23
Device(config-if)# no switchport
Device(config-if)# queuing mode sub-interface priority-propagation
Device(config-subif)# end
```

This is an example of how to configure the hierarchical QoS policy on a subinterface:

```
Device# configure terminal
Device(config)# class-map match-all traffic-class7
Device(config-cmap)# match traffic-class 7
Device(config-cmap)# exit
Device(config)# policy-map type queueing child
Device(config-pmap)# class traffic-class7
Device(config-pmap-c)# shape average 1000000
Device(config-pmap-c)# priority level 1
Device(config-pmap-c)# exit
Device(config-pmap)# exit
Device(config)# policy-map type queueing parent
Device(config-pmap)# class class-default
Device(config-pmap-c)# shape average 2000000000
Device(config-pmap-c)# service-policy child
Device(config-pmap-c)# exit
Device(config-pmap)# exit
Device(config)# interface HundredGigE1/0/5
Device(config-if)# no switchport
Device(config-pmap)# exit
Device(config)# interface HundredGigE1/0/5.1
Device(config-if)# encapsulation dot1Q 11
Device(config-if)# service-policy type queueing output parent
Device(config-if)# end
```

This is an example of how to configure the hierarchical QoS bandwidth remaining ratio on subinterface:

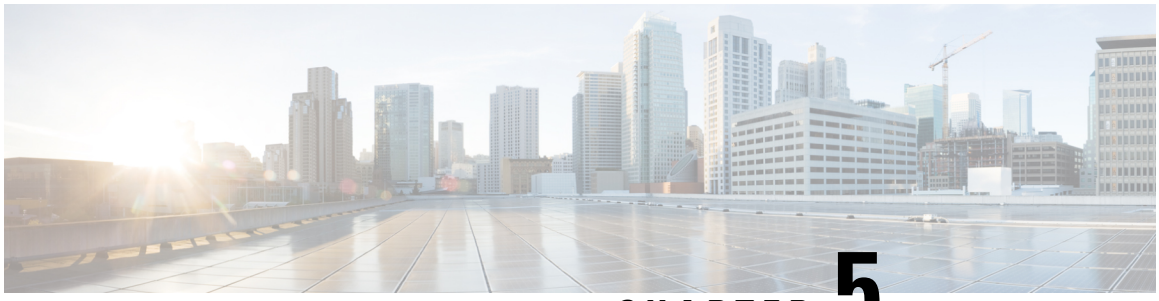
```
Device# configure terminal
Device(config)# class-map match-all traffic-class7
Device(config-cmap)# match traffic-class 7
Device(config-cmap)# exit
Device(config)# policy-map type queueing child
Device(config-pmap)# class traffic-class7
Device(config-pmap-c)# shape average 1000000
Device(config-pmap-c)# priority level 1
Device(config-pmap-c)# exit
Device(config-pmap)# exit
Device(config)# policy-map type queueing parent_ratio_5
Device(config-pmap)# class class-default
Device(config-pmap-c)# bandwidth remaining ratio 5
Device(config-pmap-c)# service-policy child
Device(config-pmap-c)# exit
Device(config-pmap)# exit
Device(config)# policy-map type queueing parent_ratio_10
Device(config-pmap)# class class-default
Device(config-pmap-c)# bandwidth remaining ratio 10
Device(config-pmap-c)# service-policy child
Device(config-pmap-c)# exit
Device(config-pmap)# exit
Device(config)# interface HundredGigE1/0/5
Device(config-if)# no switchport
Device(config-pmap)# exit
```

```
Device(config)# interface HundredGigE1/0/5.1
Device(config-if)# encapsulation dot1Q 11
Device(config-if)# service-policy type queueing output parent_ratio_5
Device(config-if)# exit
Device(config)# interface HundredGigE1/0/5.2
Device(config-if)# encapsulation dot1Q 15
Device(config-if)# service-policy type queueing output parent_ratio_10
Device(config-if)# end
```

Monitor Layer 3 subinterface queuing

These commands can be used to monitor Layer 3 subinterface queuing configuration on the device.

Command	Description
show policy-map type queueing interface *interface-id[.subinterface]*	Displays the runtime representation and statistics of all the queueing policies configured on the device.
show running interface *interface-id[.subinterface]*	Displays the configured interface and values under it.
show running policy-map *name*	Displays a list of the policy maps along with traffic class information.



CHAPTER 5

MPLS QoS

This document provides detailed understanding about MPLS QoS and its configuration.

- Feature history for MPLS QoS, on page 161
- MPLS QoS, on page 161
- MPLS experimental field, on page 162
- Benefits of MPLS EXP classification and marking, on page 163
- Prerequisites for MPLS QoS, on page 163
- Restrictions for MPLS QoS, on page 163
- Configure MPLS QoS, on page 164
- Configuration examples, on page 168

Feature history for MPLS QoS

This table provides release and platform support information for the features explained in this module.

These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Release	Feature name and description	Supported platform
Cisco IOS XE 26.2.1ea	MPLS QoS: MPLS QoS feature support has been introduced.	Cisco C9550 Series Smart Switches
Cisco IOS XE 17.18.2	MPLS QoS: MPLS QoS feature allows you to classify, mark, and queue network traffic by modifying the MPLS experimental bits field.	Cisco C9350 Series Smart Switches Cisco C9610 Series Smart Switches

MPLS QoS

The MPLS QoS functionality enables network administrators to provide differentiated services across an MPLS network. Network administrators can satisfy a wide range of networking requirements by specifying the class of service (CoS) applicable to each transmitted IP packet. Different classes of service are established for IP packets by setting the IP precedence bit in the packet header.

Classifications, remarkings, and queuings on MPLS networks are performed over MPLS EXP bits. In the MPLS network, the packets are differentiated by the MPLS EXP field marking, and are treated accordingly, depending on the weighted early random detection (WRED) configuration.

The MPLS EXP field in an MPLS packet allows you to:

- Classify traffic: The classification process selects the traffic to be marked. Classification accomplishes this by partitioning traffic into multiple priority levels, or classes of service. Traffic classification is the primary component of class-based QoS provisioning.
- Police and mark traffic: Policing causes traffic that exceeds the configured rate to be discarded or marked to a different drop level. Marking traffic is a way to identify packet flows to differentiate them. Packet marking allows you to partition your network into multiple priority levels or classes of service.
- Queueing: Queueing prevents congestion with priority level queueing, WTD, scheduling, shaping, and WRED features.

The system supports HQoS on MPLS QoS, as same as standard QoS.

MPLS QoS and standard QoS share the same set of configuration restrictions.

MPLS experimental field

The MPLS experimental bits (EXP) field is a 3-bit field in the MPLS header used to define the QoS treatment (per-hop behavior) given to a packet by a node. In an IP network, the DiffServ Code Point (DSCP) (a 6-bit field) defines a class and drop precedence. The EXP bits can be used to carry some of the information encoded in the IP DSCP and can also be used to encode the dropping precedence.

Cisco IOS Software defaults to copying the three most significant bits of the DSCP or the IP precedence of the IP packet to the EXP field in the MPLS header. This action happens when the MPLS header is initially imposed on the IP packet. However, you can also set the EXP field by defining a mapping between the DSCP or IP precedence and the EXP bits. This mapping is configured using the **set mpls experimental** or **police** commands.



Note A policy map configured with **set ip dscp** is not supported on the provider edge device because the policy action for MPLS label imposition node should be based on **set mpls experimental imposition** value. However, a policy map with action **set ip dscp** is supported when both the ingress and egress interfaces are Layer 3 ports.

You can perform MPLS EXP marking operations using table-maps. It is recommended to assign QoS-group to a different class of traffic in ingress policy and translate QoS-group to DSCP and EXP markings in egress policy using table-map.

Service providers may choose not to modify the value of the IP precedence field in packets transported through their network. Instead, they can use the MPLS EXP field value to classify and mark IP packets. By selecting different values for the MPLS EXP field, critical packets can be marked to ensure priority during network congestion.

WRED monitors network traffic to anticipate congestion and prevent it at common network and internetwork bottlenecks. WRED can selectively discard lower priority traffic when an interface becomes congested. This feature can also provide differentiated performance characteristics for different classes of service.

Packet transport through an MPLS network

There are two ways to transport packets through the MPLS network:

- Uniform mode: Uniform mode of transferring packets operates on one layer of QoS. The Provider Edge at ingress copies the DSCP information from the incoming IP packet into the MPLS EXP bits of the imposed labels, mapping the IP precedence bits to the MPLS EXP field. As the EXP bits travel through the core, they may or may not be modified by the intermediate devices on the network. The Provider Edge at egress copies the EXP bits to the DSCP bits of the newly exposed IP packet.
- Pipe mode: Pipe mode of transferring packets operates on two layers of QoS. An underlying QoS for the data that remains unchanged when traversing the core. A per-core QoS, which is separate from that of the underlying IP packets. The DSCP information is saved and stored as the packet travels through the MPLS network. The MPLS EXP label is applied by the PE at ingress but the IP precedence bits are not stored. At egress, the original IP precedence value is preserved.

Benefits of MPLS EXP classification and marking

The QoS EXP Matching feature enables the classification, marking, and queuing of network traffic by modifying the Multiprotocol Label Switching (MPLS) experimental bits (EXP) field. If you do not want to modify the value of the IP precedence field in packets transported through the network, use the MPLS EXP field value to classify and mark IP packets.

By choosing different values for the MPLS EXP field, you can mark critical packets so that those packets have priority if network congestion occurs.

Prerequisites for MPLS QoS

- Configure the switch as an MPLS provider edge (PE) or provider (P) router, including valid label protocol and underlying IP routing protocols.

Restrictions for MPLS QoS

- MPLS classification and marking can only occur in an operational MPLS network.
- If a packet is classified by IP type of service (ToS) or class of service (CoS) at ingress, it cannot be reclassified by MPLS EXP at egress (imposition case). However, if a packet is classified by MPLS at ingress, it can be reclassified at egress by IP ToS, CoS, or Quality of Service (QoS) group (disposition case).
- Use QoS-group to apply QoS on traffic across protocol boundaries. You can classify and assign ingress traffic to the QoS-group. Then, use the QoS-group at egress to classify and apply QoS.
- When a packet is encapsulated in MPLS, it cannot be checked for classification or marking of other protocols such as IP. Only MPLS EXP marking affects packets that are encapsulated by MPLS.
- The short pipe mode is not supported to transport packets through the MPLS network. Transport packets using either uniform mode or pipe mode.

- MPLS classification and marking are not supported on SVI. Apply the policy on either physical or port-channel interfaces.

Configure MPLS QoS

This section provides information about configuring MPLS QoS.

Classify MPLS encapsulated packets

Use the **match mpls experimental topmost** command to define traffic classes based on the packet EXP values, inside the MPLS domain. Use these classes to define service policies to mark the EXP traffic using the **police** command.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**class-map** [**match-all** | **match-any**] *class-map-name* **Example:** `Device(config)# class-map exp3`	Creates a class map to be used for matching traffic to a specified class, and enters class-map configuration mode. Enter the class map name.
Step 4	**match mpls experimental topmost** *mpls-exp-value* **Example:** `Device(config-cmap)# match mpls experimental topmost 3`	Specifies the match criteria. **Note** The **match mpls experimental topmost** command classifies traffic on the basis of the EXP value in the topmost label header.
Step 5	**end** **Example:** `Device(config-cmap)# end`	(Optional) Returns to privileged EXEC mode.

Mark MPLS EXP on the outermost label

Before you begin

Marking MPLS packets at imposition is used with ingress classification on IP ToS or CoS fields.

Note

- For IP imposition marking, the IP precedence value is copied to the MPLS EXP value by default.
- The egress policy on provider edge works with MPLS EXP class match, only if there is a remarking policy at ingress. The provider edge at ingress is an IP interface and only DSCP value is trusted by default. If you do not configure remarking policy at ingress the label for queueing is generated based on DSCP value and not MPLS EXP value. However, a transit provider router works without configuring remarking policy at ingress as the router works on MPLS interfaces.
- The **set mpls experimental imposition** command works only on packets that have new or additional MPLS labels added to them.

Perform this task to set the value of the MPLS EXP field on imposed label entries.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**policy-map** *policy-map-name* **Example:** `Device(config)# policy-map mark-up-exp-2`	Specifies the name of the policy map to be created and enters policy-map configuration mode. Enter the policy map name.
Step 4	**class** *class-map-name* **Example:** `Device(config-pmap)# class prec012`	Creates a class map to be used for matching traffic to a specified class, and enters class-map configuration mode. Enter the class map name.
Step 5	**set mpls experimental imposition** *mpls-exp-value* **Example:**	Sets the value of the MPLS EXP field on top label.

	Command or Action	Purpose
	`Device(config-pmap-c)# set mpls experimental imposition 2`	
Step 6	**end** **Example:** `Device(config-pmap-c)# end`	(Optional) Returns to privileged EXEC mode.

Mark MPLS EXP on label switched packets

Note Use the **set mpls experimental topmost** command to mark the EXP value for the outermost label of MPLS traffic. Since this marking occurs at ingress policy, ensure the egress policy includes classification based on MPLS EXP values.

Perform this task to set the MPLS EXP field on label switched packets.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**policy-map** *policy-map-name* **Example:** `Device(config)# policy-map mark-up-exp-2`	Specifies the name of the policy map to be created and enters policy-map configuration mode. Enter the policy map name.
Step 4	**class** *class-map-name* **Example:** `Device(config-pmap)# class-map exp012`	Creates a class map to be used for matching traffic to a specified class, and enters class-map configuration mode. Enter the class map name.
Step 5	**set mpls experimental topmost** *mpls-exp-value* **Example:**	Sets the MPLS EXP field value in the topmost label on the output interface.

	Command or Action	Purpose
	`Device(config-pmap-c)# set mpls experimental topmost 2`	
Step 6	**end** **Example:** `Device(config-pmap-c)# end`	(Optional) Returns to privileged EXEC mode.

Configure conditional marking

Before you begin



Note Use the **set-mpls-exp-topmost-transmit** action only on MPLS encapsulated packets. Use the **set-mpls-exp-imposition-transmit** action on any new labels that are added to the packet.

Perform this task to conditionally set the MPLS EXP field value for all imposed labels.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**policy-map** *policy-map-name* **Example:** `Device(config)# policy-map ip2tag`	Specifies the name of the policy map to be created and enters policy-map configuration mode. • Enter the policy map name.
Step 4	**class** *class-map-name* **Example:** `Device(config-pmap)# class iptcp`	Creates a class map to be used for matching traffic to a specified class, and enters policy-map class configuration mode. • Enter the class map name.

	Command or Action	Purpose
Step 5	**police cir** *bps* **bc pir** *bps* **be** **Example:** `Device(config-pmap-c)# police cir 100000000 pir 200000000`	Defines a policer for classified traffic and enters policy-map class police configuration mode.
Step 6	**conform-action transmit** **Example:** `Device(config-pmap-c-police)# conform-action transmit`	Defines the action to take on packets that conform to the values specified by the policer.
Step 7	**exceed-action set-mpls-exp-topmost-transmit exp table** *table-map-name* **Example:** `Device(config-pmap-c-police)# exceed-action set-mpls-exp-topmost-transmit exp table dscp2exp`	Defines the action to take on packets that exceed the values specified by the policer.
Step 8	**violate-action drop** **Example:** `Device(config-pmap-c-police)# violate-action drop`	Defines the action to take on packets whose rate exceeds the peak information rate (pir) and is outside the bc and be ranges. • Ensure the exceed action precedes the violate action. • In this example, if the packet rate exceeds the pir rate and is outside the bc and be ranges, the packet is dropped.
Step 9	**end** **Example:** `Device(config-pmap-c-police)# end`	(Optional) Returns to privileged EXEC mode.

Configuration examples

This section provides configuration examples for MPLS QoS.

Note MPLS traffic can be queued by classifying packets and assigning them to different queues based on their class of service (CoS) or MPLS experimental (EXP) bits.

Example: Classify MPLS encapsulated packets

Define a color of traffic for QoS enforcement

This example shows how to define a class map for the MPLS EXP Marking based on color of traffic for QoS enforcement:

```
Device(config)#class
Device(config)#class-map match-any c1
Device(config-cmap)#match mpls experimental topmost 1
Device(config-cmap)#exit
Device(config)#policy-map p1
Device(config-pmap)#class c1
Device(config-pmap-c)#police cir 2570000 ?
Device(config-pmap-c)#police cir 2570000 conform-action transmit ?
Device(config-pmap-c-police)#
Device(config-pmap-c-police)#$nform-action transmit exceed-action drop
Device(config-pmap-c-police)#exit
Device(config-pmap-c)#
```

Define an MPLS EXP class map

This example shows how to define a class map named exp3 that matches packets that contains MPLS experimental value 3:

```
Device(config)# class-map exp3
Device(config-cmap)# match mpls experimental topmost 3
Device(config-cmap)# exit
```

Define a policy map and apply the policy map to an ingress interface

This example shows how to use the class map created in the example above to define a policy map. It also shows how to apply the policy map to a physical interface for ingress traffic.

```
Device(config)# policy-map change-exp-3-to-2
Device(config-pmap)# class exp3
Device(config-pmap-c)# set mpls experimental topmost 2
Device(config-pmap)# exit
Device(config)# interface FiftyGigE1/0/2
Device(config-if)# service-policy input change-exp-3-to-2
Device(config-if)# exit
```

Example: Mark MPLS EXP on outermost label

Define an MPLS EXP imposition policy map

This example defines a policy map that sets the MPLS EXP imposition value to 2 based on the IP precedence value of the forwarded packet:

```
Device# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Device(config)# class-map prec012
Device(config-cmap)# match ip prec 0 1 2
Device(config-cmap)# exit
```

```
Device(config)# policy-map mark-up-exp-2
Device(config-pmap)# class prec012
Device(config-pmap-c)# set mpls experimental imposition 2
Device(config-pmap-c)# exit
Device(config-pmap)# exit
```

Apply the MPLS EXP imposition policy map to a main interface

This example applies a policy map to Fifty Gigabit Ethernet interface 1/0/2:

```
Device# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Device(config)# interface FiftyGigE1/0/2
Device(config-if)# service-policy input mark-up-exp-2
Device(config-if)# exit
```

Example: Mark MPLS EXP on label-switched packets

Define an MPLS EXP label switched packets policy map

This example shows how to define a policy map that sets the MPLS EXP top-most value to 2 according to the MPLS EXP value of the forwarded packet:

```
Device# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Device(config)# class-map exp012
Device(config-cmap)# match mpls experimental topmost 0 1 2
Device(config-cmap)# exit
Device(config-cmap)# policy-map mark-up-exp-2
Device(config-pmap)# class exp012
Device(config-pmap-c)# set mpls experimental topmost 2
Device(config-pmap-c)# exit
Device(config-pmap)# exit
```

Apply the MPLS EXP on label-switched packets policy map to a main interface

This example shows how to apply the policy map to a main interface:

```
Device# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Device(config)# interface FiftyGigE1/0/2
Device(config-if)# service-policy input mark-up-exp-2
Device(config-if)# exit
```

Example: Configure conditional marking

This example shows how to create a policer for the **iptcp** class, which is part of the **ip2tag** policy map, and attaches the policy map to the Fifty Gigabit Ethernet interface interface:

```
Device(config)# policy-map ip2tag
Device(config-pmap)# class iptcp
Device(config-pmap-c)# police cir 100000000 pir 200000000
Device(config-pmap-c-police)# conform-action transmit
Device(config-pmap-c-police)# exceed-action set-mpls-exp-imposition-transmit 2
```

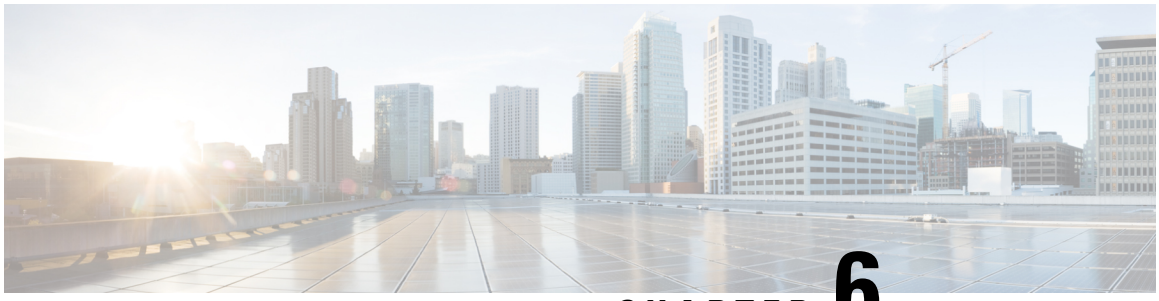
```
Device(config-pmap-c-police)# violate-action drop
Device(config-pmap-c-police)# exit
Device(config-pmap-c)# exit
Device(config-pmap)# exit
Device(config)# interface FiftyGigE1/0/2
Device(config-if)# service-policy input ip2tag
```

This is a sample output of the **sh policy-map interface** command.

```
Device(config-pmap)#interface HundredGigE1/0/1
Device(config-if)#service-policy input ip2tag
Device(config-if)#do sh policy-map interface HundredGigE1/0/1
 HundredGigE1/0/1

  Service-policy input: ip2tag

    Class-map: class-default (match-any)
      0 packets
      Match: any
```



CHAPTER 6

Application Visibility and Control QoS

This document provides detailed understanding about Application Visibility and Control (AVC) QoS and its configuration.

- Feature History for Application Visibility and Control, on page 173
- Application Visibility and Control, on page 174
- Restrictions for Application Visibility and Control, on page 175
- Supported AVC class map and policy map formats, on page 176
- Match and collect fields in a wired AVC monitor, on page 178
- NBAR2 custom applications, on page 179
- NBAR2 dynamic hitless protocol pack upgrade, on page 182
- Configure Application Visibility and Control, on page 183
- Monitor Application Visibility and Control, on page 194
- Verify Application Visibility and Control, on page 195
- Troubleshooting guidelines for Application Visibility and Control, on page 205

Feature History for Application Visibility and Control

This table provides release and platform support information for the features explained in this module.

These features are available in all the releases subsequent to the one they were introduced in, unless noted otherwise.

Table 12: Feature history for Application Visibility and Control

Release	Feature name and description	Supported platform
Cisco IOS XE 26.1.1	Application Visibility and Control QoS: The feature provides application-aware traffic management by identifying, classifying, and prioritizing network traffic on wired access ports to optimize performance.	Cisco C9350 Series Smart Switches

Release	Feature name and description	Supported platform
Cisco IOS XE 17.18.2	Application Visibility and Control Flexible NetFlow: AVC FNF is a network traffic analytics and application visibility solution that combines customizable flow data collection with advanced application recognition to enable detailed monitoring, classification, and control of network traffic for optimized performance and security.	Cisco C9350 Series Smart Switches

Application Visibility and Control

Cisco Application Visibility and Control (AVC) is a network management solution that

- classifies applications using deep packet inspection (DPI) techniques
- enables application-aware network policies on wired access ports, and
- provides detailed client, server, and application-level traffic statistics.

Deep packet inspection (DPI): AVC uses Network-Based Application Recognition (NBAR2) to recognize applications through Layers 4 to 7. NBAR2 can be activated either explicitly on the interface by enabling protocol-discovery or implicitly by attaching a QoS policy that contains **match protocol** classifier.

AVC QoS is an application-aware traffic management feature that

- leverages the granular visibility of applications that Cisco AVC solution provides
- identifies, classifies, and prioritizes network traffic based on the specific application, and
- allows network administrators to drop, mark, or rate-limit multiple applications to optimize network performance.

Benefits of Application Visibility and Control

- Enhanced intelligence: Evolves branch and campus solutions from strictly packet and connection-based management to application-aware, application-intelligent systems.
- Granular visibility: Provides detailed statistics per interface, allowing for better insight into network usage patterns.

  You can configure wired AVC Flexible NetFlow (FNF) on an interface to provide client, server and application statistics per interface.
- Flexible deployment: Supports configuration on wired access ports for standalone switches as well as for a switch stack.

- Performance monitoring: Offers traffic monitoring capabilities similar to Easy Performance Monitor (ezPM) to track application performance and client-server interactions.

  The record is similar to **application-client-server-stats** traffic monitor which is available in **application-statistics** and **application-performance** profiles in ezPM.

Restrictions for Application Visibility and Control

Interface and port limitations

- Allows configuration only on wired physical ports
- Supports only Layer 2 access, trunk ports, and Layer 3 routed ports

Does not support policy configuration on these:

- virtual interfaces like VLAN and other logical interfaces
- port-channel member ports and virtual interfaces like SVIs or sub-interfaces, and
- management port (Gig 0/0).

Traffic and protocol support

- Does not support multicast traffic
- Supports only IPv4/IPv6 unicast TCP and UDP traffic
- Supports only IPv4/IPv6 unicast TCP and UDP when using `match application name` in a Netflow record
- Routed DHCP traffic and egress copies of L2 unknown-unicast flooded traffic are not learnt. Whereas, IPv6 non-initial fragment filtering packets are learnt.

Co-existence and operational restrictions

- Supports a maximum of four flow monitors with different flow records per interface per direction per traffic type
- Does not support configuring NBAR and transmit (Tx) Switched Port Analyzer (SPAN) on the same interface
- Does not support configuring NBAR and ACL logging on the same switch
- You cannot configure protocol-discovery, application-based QoS, and wired AVC FNF together at the same time on the same interface with the non-application-based FNF. However, these wired AVC features can be configured with each other. For example, you can configure protocol-discovery, application-based QoS, and wired AVC FNF on the same interface at the same time.
- Does not support Encrypted Traffic Analytics (ETA)
- Does not support AVC on uplink interfaces
- When exporting AVC flow records with an external collector, the interface input field is not included as a match field, so the information from all interfaces is aggregated into the same record.

QoS policy and classification restrictions

- Supports AVC QoS policy only in the ingress direction.
- Allows NBAR2-based match criteria, **match protocol**, only with the marking or policing actions. Does not allow NBAR2 match criteria in a policy that has queuing features configured.
- Limit the number of concurrent protocols to 255 across all policies. The hardware limitation is 8 bit.
- Allows only one of the NBAR-based QoS mechanisms—protocol-based or attributes-based—to be attached to any port at the same time
- Use only the `traffic-class` and `business-relevance` attributes for attribute-based QoS
- Account for a slight delay in QoS classification, as application identification occurs offline while initial packets of the flow are forwarded
- For egress traffic, AVC or FNF flow learning occurs only when the packet matches the FNF or AVC learn path. If an egress acl deny policy drops the packet before learning or mirroring, no FNF flow record is created and no AVC or NBAR classification is performed for that flow.

Management and UI restrictions

- Use the Command Line Interface (CLI) for all application control tasks, as it is not supported on the Web UI.
- To manage and check wired AVC traffic on the Web UI, you must first configure **ip http authentication local** and **ip nbar http-service** commands using the CLI.

Performance and scalability

- Supports a maximum of four flow monitors with different flow records per interface per direction per traffic type
- Supports 8K AVC flows across IPv4/IPv6 ingress flows per ASIC

Supported AVC class map and policy map formats

This section describes the supported AVC class maps and policy map formats.

Supported AVC class map format

Class map format	Class map example	Direction
match protocol *protocol name*	`class-map match-any NBAR-VOICE` `match protocol ms-lync-audio`	ingress

Combination filters	`class-map match-any NBAR-VOICE`
	`match protocol ms-lync-audio`
	`match dscp ef`
		ingress

Supported AVC policy format

Policy Format	QoS Action
Ingress policy based on match protocol filter	Mark and police

The table describes the detailed AVC policy format with an example.

AVC Policy Format	AVC Policy Example	Direction

Basic set	`policy-map MARKING-IN`
	`class NBAR-MM_CONFERENCING`
	`set dscp af41`
		ingress

Basic police	`policy-map POLICING-IN`
	`class NBAR-MM_CONFERENCING`
	`police cir 600000`
	`set dscp af41`
		ingress

Basic set and police	`policy-map webex-policy`
	` class webex-class`
	` set dscp ef`
	` police 5000000`
		ingress

Multiple set and police including default	`policy-map webex-policy`
	` class webex-class`
	` set dscp af31`
	` police 4000000`
	` class class-webex-category`
	` set dscp ef`
	`  police 6000000`
	` class class-default`
	` set dscp <>`
		ingress

Hierarchical police	`policy-map webex-policy`
	` class webex-class`
	` police 5000000`
	` service-policy client-in-police-only`
	
	` policy-map client-in-police-only`
	` class webex-class`
	` police 100000`
	` class class-webex-category`
	` set dscp ef`
	`  police 200000`
		ingress

Hierarchical set and police	`policy-map webex-policy`
	` class class-default`
	` police 1500000`
	` service policy client-up-child`
	` policy-map client-up-child`
	` class webex-class`
	` police 100000`
	` set dscp ef`
	` class class-webex-category`
	` police 200000`
	` set dscp af31`
		ingress

Match and collect fields in a wired AVC monitor

You can use any combination of the match and collect fields listed here in a wired AVC monitor.

Application and flow information

- application name
- flow direction
- connection initiator
- flow observation point

IP address information

IPv4:

- IPv4 version
- IPv4 protocol
- IPv4 source address
- IPv4 destination address
- Connection client IPv4 address
- Connection server IPv4 address

IPv6:

- IPv6 version
- IPv6 protocol
- IPv6 source address
- IPv6 destination address
- connection client IPv6 address
- connection server IPv6 address

Transport layer information

- transport TCP flags
- transport source-port
- transport destination-port
- connection client transport port
- connection server transport port

Interface information

- interface input
- interface output

The fields `interface input` and `interface output` can each be used as a match field in a record. But both the fields cannot be used as match fields in the same record. You can used both the fields as collect fields or one as a collect field and one as a match field in the same record.

Supported collect only fields

- connection client counter packets long
- connection client counter bytes network long
- connection server counter packets long
- connection server counter bytes network long
- counter bytes long
- counter packets long
- timestamp absolute first
- timestamp absolute last
- connection new-connections

Extracted fields

- application dns domain-name
- application http host
- application ssl common-name

NBAR2 custom applications

NBAR2 supports the use of custom protocols to identify custom applications. Custom protocols support protocols and applications that NBAR2 does not currently support.

Local application categories

In every deployment, there are local and specific applications which are not covered by the NBAR2 protocol pack provided by Cisco. Local applications are mainly categorized as:

- specific applications to an organization, and
- applications specific to a geography

Application customization types

NBAR2 provides a way to manually customize local applications. Custom applications take precedence over built-in protocols. For each custom protocol, user can define a selector ID that can be used for reporting purposes.

There are various types of application customization:

- Generic protocol customization:
  - HTTP
  - SSL
  - DNS
- Composite customization: customization based on multiple underlying protocols
- Layer3 or Layer4 customization:
  - IPv4 address
  - DSCP values
  - TCP/UDP ports
  - Flow source or destination direction
- Byte Offset: customization based on specific byte values in the payload

Customize local applications using NBAR2

Use this procedure to manually customize local applications using NBAR2.

You can manually customize applications using the command **ip nbar custom** *myappname* in global configuration mode.

Procedure

Step 1 Customize general protocols such as HTTP, SSL, or DNS.

Example:

Custom application called MYHTTP using the HTTP host "*mydomain.com" with Selector ID 10:

```
Device# configure terminal
Device(config)# ip nbar custom MYHTTP http host *mydomain.com id 10
```

Customization can be done for SSL encrypted traffic using information extracted from the SSL Server Name Indication (SNI) or Common Name (CN).

Custom application called MYSSL using SSL unique-name "mydomain.com" with selector ID 11:

```
DDevice# configure terminal
Device(config)#ip nbar custom MYSSL ssl unique-name *mydomain.com id 11
```

Custom application called MYDNS using the DNS domain name “mydomain.com” with selector ID 12:

```
Device# configure terminal
Device(config)# ip nbar custom MYDNS dns domain-name *mydomain.com id 12
```

NBAR2 examines DNS request and response traffic, and can correlate the DNS response to an application. The IP address returned from the DNS response is cached and used for later packet flows associated with that specific application.

To extend an existing application, use the command **ip nbar custom** *application-name* **dns domain-nam** *domain-name* **extends** *existing-application* .

For more information on DNS-based customization, see http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_nbar/configuration/xe-3s/asr1000/qos-nbar-xe-3s-asr-1000-book/nbar-custapp-dns-xe.html.

Step 2 Customize composite or multiple underlying protocols.

Example:

Custom application called MYDOMAIN using HTTP, SSL or DNS domain name “mydomain.com” with selector ID 13:

```
Device# configure terminal
Device(config)# ip nbar custom MYDOMAIN composite server-name *mydomain.com id 13
```

Step 3 Customize Layer3 or Layer4.

Example:

Custom application called LAYER4CUSTOM matching IP addresses 10.56.1.10 and 10.56.1.11, TCP and DSCP ef with selector ID 14:

```
Device# configure terminal
Device(config)# ip nbar custom LAYER4CUSTOM transport tcp id 14
Device(config-custom)# ip address 10.56.1.10 10.56.1.11
Device(config-custom)# dscp ef
```

L3/L4 customization is based on the packet tuple and is always matched on the first packet of a flow.

Step 4 Monitor custom applications.

Example:

```
Device# show ip nbar  protocol-id | inc Custom
LAYER4CUSTOM             14            Custom
MYDNS                    12            Custom
MYDOMAIN                 13            Custom
MYHTTP                   10            Custom
MYSSL                    11            Custom

Device# show ip nbar  protocol-id MYSSL
Protocol Name              id            type
-----------------------------------------------
MYSSL                    11            Custom
```

NBAR2 dynamic hitless protocol pack upgrade

Protocol packs are software packages that update the NBAR2 protocol support on a device without replacing the Cisco software on the device. A protocol pack contains information on applications officially supported by NBAR2 which are compiled and packed together. For each application, the protocol-pack includes information on application signatures and application attributes. Each software release has a built-in protocol-pack bundled with it.

Protocol pack features

Protocol packs provide these features:

- They are easy and fast to load.
- They are easy to upgrade to a higher version protocol pack or revert to a lower version protocol pack.
- They do not require the switch to be reloaded.



Warning When using switch stacking, ensure that each switch has the same Protocol Pack file loaded. If you execute the **ip nbar protocol-pack flash** *protocol-pack-file* command on the primary switch in the stack, any switch in the stack that does not have the file loaded will be reloaded due to a configuration mismatch.

NBAR2 protocol packs are available for download on Cisco Software Center from this URL: https://software.cisco.com/download/home.

Load the NBAR2 protocol pack

Use this procedure to load and activate the NBAR2 protocol pack to expand application recognition capabilities beyond the built-in protocols.

NBAR2 protocol packs contain updated protocol definitions and signatures that enhance the ability to identify and classify network applications. Loading a protocol pack updates the device with the latest application recognition patterns.

Procedure

Step 1 Load the protocol pack.

Example:

```
Device> enable
Device# configure terminal
Device(config)# ip nbar protocol-pack flash:defProtoPack
Device(config)# exit
```

Here, flash:defProtoPack is the *protocol-pack-location*.

Use the **force** keyword to specify and load a protocol pack of a lower version, which is different from the base protocol pack version. This also removes the configuration that is not supported by the current protocol pack on the switch.

For reverting to the built-in protocol pack, use this command:

Example:

```
Device(config)# default ip nbar protocol-pack
Device(config)# exit
```

Step 2 Display the protocol pack information and verify the loaded protocol pack version, publisher, and other details.

Example:

```
Device# show ip nbar protocol-pack active
```

- **protocol-pack** keyword: displays information about the specified protocol pack
- **active** keyword: displays active protocol pack information
- **detail** keyword: displays detailed protocol pack information

The NBAR2 protocol pack is successfully loaded and activated, providing enhanced application recognition capabilities with the latest protocol definitions.

Configure Application Visibility and Control

Use this procedure to monitor and control network applications by implementing visibility and control mechanisms on wired network interfaces.

Protocol-Discovery, application-based QoS, and application-based flexibile Netflow (FNF) or wired AVC FNF are all independent features. They can be configured independently or together on the same interface at the same time.

Procedure

Step 1 Enable application recognition on an interface, on page 183
Step 2 Configure Application Visibility and Control QoS, on page 184.
Step 3 Configure application-based flexible Netflow.

Enable application recognition on an interface

Use this procedure to enable application recognition on a network interface using the NBAR2 engine.

Procedure

Step 1 Specify the interface for which you are enabling protocol discovery and enter the interface configuration mode.

Example:

```
Device# configure terminal
Device(config)# interface gigabitethernet 1/0/1
```

Step 2 Enable application recognition on the interface by activating NBAR2 engine.

Example:

```
Device(config-if)# ip nbar protocol-discovery
Device(config-if)# end
```

Application recognition is now enabled on the specified interface, allowing the NBAR2 engine to discover and classify application protocols on network traffic.

Configure Application Visibility and Control QoS

Use this procedure to configure application-based QoS on a switch port interface to control traffic flow and prioritization.

QoS policies must be applied to switch port interfaces to enforce traffic management rules. This configuration enables the switch to classify, mark, and prioritize traffic according to the defined policy map.

Policy maps are used to define QoS actions that are applied to classified traffic. You can configure policing to limit traffic rates and set DSCP or CoS values to mark packets for appropriate treatment by network devices.

The class map is created and configured with the specified match criteria for traffic classification.

Procedure

Step 1 Create a class map with match protocol filters.

Example:

```
Device# configure terminal
Device(config)# class-map webex-class
Device(config-cmap)# match protocol webex-media
Device(config-cmap)# end
```

Example:

This example shows how to match on attributes:

```
Device# configure terminal
Device(config)# class-map match-all audio
Device(config-cmap)# match protocol attribute traffic-class voip-telephony
```

```
Device(config-cmap)# match protocol attribute business-relevance business-relevant
Device(config-cmap)# end
```

You need to create a class map before configuring any match protocol filter. The QoS actions such as marking and policing can be applied to the traffic. The AVC match protocol filters are applied to the wired access ports. For more information about the protocols that are supported, refer http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_nbar/prot_lib/config_library/nbar-prot-pack-library.html.

Step 2 Create a policy map to define traffic classification and apply QoS actions such as policing and marking to network traffic.

a) Create a policy map.

Example:

```
Device# configure terminal
Device(config)# policy-map webex-policy
```

By default, no policy maps are defined.

The default behavior of a policy map is to set the DSCP to 0 if the packet is an IP packet and to set the CoS to 0 if the packet is tagged. No policing is performed.

To delete an existing policy map, use the **no policy-map** *policy-map-name* global configuration command.

b) Define a traffic classification.

Example:

```
Device(config-pmap)#  class webex-class
```

By default, no policy map and class maps are defined.

If a traffic class has already been defined by using the **class-map** global configuration command, specify its name for *class-map-name* in this command.

A **class-default** traffic class is predefined and can be added to any policy. It is always placed at the end of a policy map. With an implied **match any** is included in the **class-default** class, all packets that have not already matched the other traffic classes will match **class-default**.

To delete an existing class map, use the **no class** *class-map-name* policy-map configuration command.

c) Define a policer for the classified traffic.

Example:

```
Device(config-pmap-c)# police 100000 80000
```

By default, no policer is defined.

Specify these values:

- For *rate-bps,* specify an average traffic rate in bits per second (b/s). The range is 8000 to 10000000000.
- For *burst-byte,* specify the normal burst size in bytes. The range is 1000 to 512000000.

d) Classify IP traffic by setting a new value in the packet.

Example:

```
Device(config-pmap-c)# set dscp 45
Device(config-pmap-c)# end
```

For **dscp** *new-dscp*, enter a new DSCP value to be assigned to the classified traffic. The range is 0 to 63.

Step 3 Apply the QoS policy to a switch port to enforce traffic marking, shaping, and prioritization rules on incoming traffic.

Example:

```
Device# configure terminal
Device(config)#  interface Gigabitethernet 1/0/1
Device(config-if)# service-policy input webex-policy
Device(config-if)# end
```

Configure wired AVC flexible Netflow

Use this procedure to collect and export application-level statistics.

Procedure

Step 1 Create a flow record

a) Create a bidirectional flow record, on page 186
b) Create directional flow records, on page 188
c) Create a DNS flow record, on page 190

Wired AVC flexible Netflow (FNF) supports two types of predefined flow records—legacy bidirectional flow records and directional flow records (ingress and egress). A total of four different predefined flow records, two bidirectional flow records and two directional flow records, can be configured and associated with a flow monitor. The legacy bidirectional records are client/server application statistics records, and the new directional records are application-stats for input/output.

Step 2 Create a flow exporter, on page 191.

Step 3 Create a flow monitor, on page 192.

Step 4 Associate flow monitor to an interface, on page 194.

Create a flow record

Create a bidirectional flow record

Use this procedure to create a bidirectional flow record that captures traffic flow information including application names, client and server details, and connection statistics for AVC analysis.

Flow records define the key fields and values that are collected from network traffic. A bidirectional flow record captures data from both directions of a connection, providing comprehensive visibility into application traffic patterns and usage statistics.

Wired AVC flexible Netflow supports two types of predefined flow records—legacy bidirectional flow records and directional flow records (ingress and egress). A total of four different predefined flow records, two bidirectional flow records and two directional flow records, can be configured and associated with a flow

monitor. The legacy bidirectional records are client/server application statistics records, and the new directional records are application-stats for input/output.

Procedure

Step 1 Create a flow record along with a description.

Example:

```
Device(config)# flow record fr-wdavc-1
Device(config-flow-record)# description fr-wdavc-1
```

Step 2 Specify the respective match criteria.

Example:

```
Device(config-flow-record)# match ipv4 version
Device(config-flow-record)# match ipv4 protocol
Device(config-flow-record)# match application name
Device(config-flow-record)# match connection client ipv4 address
Device(config-flow-record)# match connection server ipv4 address
Device(config-flow-record)# match connection server transport port
Device(config-flow-record)# match flow observation point
```

Note
Configuring **match application name** is mandatory for AVC support, as this allows the flow to be matched against the application.

The client is the flow initiator and the server is the flow responder.

The **match flow observation point** specifies a match to the observation point ID for flow observation metrics.

Step 3 Specify the collection parameters.

Example:

```
Device(config-flow-record)# collect flow direction
Device(config-flow-record)# collect connection initiator
Device(config-flow-record)# collect connection new-connections
Device(config-flow-record)# collect connection client counter packets long
Device(config-flow-record)# collect connection client counter bytes network long
Device(config-flow-record)# collect connection server counter packets long
Device(config-flow-record)# collect connection server counter bytes network long
Device(config-flow-record)# collect timestamp absolute first
Device(config-flow-record)# collect timestamp absolute last
Device(config-flow-record)# end
```

Various collection parameters include:

- **flow direction**: specifies to collect the direction—ingress or egress—of the relevant side—initiator or responder— of the bi-directional flow that is specified by the **initiator** keyword in the **collect connection initiator** command. Depending on the value specified by the **initiator** keyword, the **flow direction** keyword takes one of these values:
  - 0x01 = Ingress Flow
  - 0x02 = Egress Flow

When the connection side is set to **initiator**, the flow direction is specified from the initiator side of the flow. When the connection side is set to **responder**, the flow direction is specified from the responder side of the flow. For wired AVC, the connection side is always set to **initiator**.

- **connection initiator**: Specifies to collect the side of the flow—Initiator or Responder—relevant to the direction of the flow specified by the **collect flow direction** command. The **initiator** keyword provides tthis information about the direction of the flow:
  - 0x01 = Initiator - the flow source is the initiator of the connection
- **connection new-connections**: specifies to collect the number of connection initiations observed.
- **connection client counter packets long**: specifies to collect the number of packets sent by the client.
- **connection client counter bytes network long**: specifies to collect the total number of bytes transmitted by the client.
- **connection server counter packets long**: specifies to collect the number of packets sent by the server.
- **connection server counter bytes network long**: specifies to collect the total number of bytes transmitted by the server.
- **timestamp absolute first**: specifies to collect the time, in milliseconds, when the first packet was seen in the flow.
- **timestamp absolute last**: specifies to collect the time, in milliseconds, when the most recent packet was seen in the flow.

Step 4 Verify information about all the flow records.

Example:

```
Device# show flow record
```

What to do next

Similarly, configure the ingress and egress directional flow records and DNS flow record.

Create directional flow records

Use this procedure to create separate directional flow records—to capture and analyze ingress and egress network traffic patterns—with application identification capabilities.

You can create separate flow record for ingress and egress traffic by specifying a match to the input or output interface, respectively. Also specify to collect from the output or input interface, respectively.

Procedure

Step 1 Create a flow record along with a description.

Example:

```
Device(config)# flow record fr-wdavc-3
Device(config-flow-record)# description flow-record-3
```

Step 2 Specify the respective match criteria.

Example:

```
Device(config-flow-record)# match ipv4 version
Device(config-flow-record)# match ipv4 protocol
Device(config-flow-record)# match ipv4 source address
Device(config-flow-record)# match ipv4 destination address
Device(config-flow-record)# match transport source-port
Device(config-flow-record)# match transport destination-port
Device(config-flow-record)# match interface input
Device(config-flow-record)# match application name
```

Note
Configuring **match application name** is mandatory for AVC support, as this allows the flow to be matched against the application.

Use **match interface input** for creating ingress flow record and **match interface output** for creating egress flow record.

Step 3 Specify the collection parameters.

Example:

```
Device(config-flow-record)# collect interface output
Device(config-flow-record)# collect counter bytes long
Device(config-flow-record)# collect counter packets long
Device(config-flow-record)# collect timestamp absolute first
Device(config-flow-record)# collect timestamp absolute last
Device(config-flow-record)# end
```

For creating ingress flow record, specify **collect interface output** and for egress flow record, specify **collect interface input**.

Various collection parameters include:

- **collect interface output**: specifies to collect the output interface from the flows.

  collect interface input: specifies to collect the input interface from the flows.

- **collect counter bytes long**: specifies to collect the number of bytes in a flow.
- **collect counter packets long**: specifies to collect the number of packets in a flow.
- **collect timestamp absolute first**: specifies to collect the time, in milliseconds, when the first packet was seen in the flow.
- **collect timestamp absolute last**: specifies to collect the time, in milliseconds, when the most recent packet was seen in the flow.

Step 4 Verify information about all the flow records.

Example:

```
Device# show flow record
```

Create a DNS flow record

Use this procedure to create a flow record specifically designed to monitor DNS traffic, collecting domain names and connection metrics for network analysis and security purposes.

DNS flow records are used to track domain name resolution patterns and collect detailed statistics about DNS connections. This configuration is essential for network monitoring, security analysis, and application visibility and control (AVC) implementations.

Procedure

Step 1 Create a flow record along with a description.

Example:

```
Device(config)# flow record fr-wdavc-5
Device(config-flow-record)# description flow-record-5
```

Step 2 Specify the respective match criteria.

Example:

```
Device(config-flow-record)# match ipv4 version
Device(config-flow-record)# match ipv4 protocol
Device(config-flow-record)# match application name
Device(config-flow-record)# match connection client ipv4 address
Device(config-flow-record)# match connection client transport port
Device(config-flow-record)# match connection server ipv4 address
Device(config-flow-record)# match connection server transport port
```

Note
Configuring **match application name** is mandatory for AVC support, as this allows the flow to be matched against the application.

Step 3 Specify the collection parameters.

Example:

```
Device(config-flow-record)# collect flow direction
Device(config-flow-record)# collect connection initiator
Device(config-flow-record)# collect connection new-connections
Device(config-flow-record)# collect connection client counter packets long
Device(config-flow-record)# collect connection client counter bytes network long
Device(config-flow-record)# collect connection server counter packets long
Device(config-flow-record)# collect connection server counter bytes network long
Device(config-flow-record)# collect timestamp absolute first
Device(config-flow-record)# collect timestamp absolute last
Device(config-flow-record)# end
```

Various collection parameters include:

- **flow direction**: specifies to collect the direction—ingress or egress—of the relevant side—initiator or responder— of the bi-directional flow that is specified by the **initiator** keyword in the **collect connection initiator** command. Depending on the value specified by the **initiator** keyword, the **flow direction** keyword takes one of these values:
  - 0x01 = Ingress Flow

- 0x02 = Egress Flow

When the connection side is set to **initiator**, the flow direction is specified from the initiator side of the flow. When the connection side is set to **responder**, the flow direction is specified from the responder side of the flow. For wired AVC, the connection side is always set to **initiator**.

- **connection initiator**: Specifies to collect the side of the flow—Initiator or Responder—relevant to the direction of the flow specified by the **collect flow direction** command. The **initiator** keyword provides tthis information about the direction of the flow:
  - 0x01 = Initiator - the flow source is the initiator of the connection
- **connection new-connections**: specifies to collect the number of connection initiations observed.
- **connection client counter packets long**: specifies to collect the number of packets sent by the client.
- **connection client counter bytes network long**: specifies to collect the total number of bytes transmitted by the client.
- **connection server counter packets long**: specifies to collect the number of packets sent by the server.
- **connection server counter bytes network long**: specifies to collect the total number of bytes transmitted by the server.
- **timestamp absolute first**: specifies to collect the time, in milliseconds, when the first packet was seen in the flow.
- **timestamp absolute last**: specifies to collect the time, in milliseconds, when the most recent packet was seen in the flow.

Step 4 Configures the use of the DNS Domain-Name as a Collect field for a DNS flow record.

Example:

```
Device(config-flow-record)# collect application dns domain-name
Device(config)# end
```

Step 5 Verify information about all the flow records.

Example:

```
Device# show flow record
```

Create a flow exporter

Use this procedure to create a flow exporter to define how flow data is exported to a collector system.

Flow exporters are used to send flow monitoring data to external systems for analysis. You need to specify the destination and configure export parameters to establish proper data collection.

Procedure

Step 1 Specify a name and description for the flow exporter.

Example:

```
Device# configure terminal
Device(config)# flow exporter flow-exporter-1
Device(config-flow-exporter)# description flow-exporter-1
```

Step 2 Specify the destination—hostname, IPv4 or IPv6 address—of the system to which the exporter sends data.

Example:

```
Device(config-flow-exporter)# destination 10.10.1.1
```

Step 3 (Optional) Configure the application table option for the flow exporter.

Example:

```
Device(config-flow-exporter)# option application-table timeout 500
Device(config)# end
```

The timeout option configures the resend time in seconds for the flow exporter. The valid range is from 1 to 86400 seconds.

Step 4 (Optional) Verify flow exporter configuration.

a) Display information about all the flow exporters.

Example:

```
Device# show flow exporter
```

b) Display flow exporter statistics.

Example:

```
Device# show flow exporter statistics
```

The flow exporter is now configured and ready to send flow data to the specified destination system for monitoring and analysis.

Create a flow monitor

Use this procedure to create a flow monitor to collect, analyze, and export network traffic flow data for monitoring and troubleshooting purposes.

You can create a flow monitor and associate it with a flow record. Flow monitors are used to collect network traffic statistics and can be configured with various cache parameters and export options.

Procedure

Step 1 Create a flow monitor and specify a description.

Example:

```
Device# configure terminal
Device(config)# flow monitor flow-monitor-1
Device(config-flow-monitor)# description flow-monitor-1
```

Step 2 Associate the flow monitor with a flow record that was created previously.

Example:

```
Device(config-flow-monitor)# record flow-record-1
```

Step 3 Associate the flow monitor with a flow exporter that was created previously.

Example:

```
Device(config-flow-monitor)# exporter flow-exporter-1
```

Step 4 (Optional) Configure flow cache parameters.

Example:

```
Device(config-flow-monitor)# cache type normal
Device(config-flow-monitor)# cache timeout active 1800
Device(config-flow-monitor)# cache timeout inactive 200
Device(config)# end
```

The entries option specifies the maximum number of flow entries in the flow cache in the range from 16 to 65536. Only normal cache type is supported.

Step 5 (Optional) Verify flow monitor configuration.

a) Display information about all the flow monitors.

Example:

```
Device# show flow monitor
```

b) Display information about the specified wired AVC flow monitor.

Example:

```
Device# show flow monitor flow-monitor-1
```

c) Display statistics for wired AVC flow monitor.

Example:

```
Device# show flow monitor flow-monitor-1 statistics
```

Step 6 Clear the statistics of the specified flow monitor.

Example:

```
Device# clear flow monitor flow-monitor-1 statistics
```

Use the **show flow monitor flow-monitor-1 statistics** command after using the **clear flow monitor flow-monitor-1 statistics** to verify that all the statistics have been reset.

Step 7 Display flow cache contents.

- In tabular format:

  ```
  Device# show flow monitor flow-monitor-1 cache format table
  ```

- In similar format as the flow record:

  ```
  Device# show flow monitor flow-monitor-1 cache format record
  ```

- In CSV format:

```
Device# show flow monitor flow-monitor-1 cache format csv
```

What to do next

Associate flow monitor to an interface.

Associate flow monitor to an interface

Use this procedure to associate a flow monitor to an interface to enable packet flow monitoring and analysis.

You can attach two different wired AVC monitors with different predefined records to an interface at the same time. This configuration enables flow monitoring for network traffic analysis and application visibility.

Procedure

Step 1 Specify the interface and enter the interface configuration mode.

Example:

```
Device(config)# interface Gigabitethernet 1/0/1
```

Step 2 Associate a flow monitor to the interface for input or output packets, or both.

Example:

```
Device(config-if)# ip flow monitor flow-monitor-1 input
```

The available direction options are: **input** and **output**.

Monitor Application Visibility and Control

Use this procedure to monitor and verify application visibility statistics, flows, and policy configurations to ensure proper Application Visibility and Control (AVC) operation.

AVC provides network administrators with detailed insights into application usage and performance. These monitoring commands help verify AVC functionality and troubleshoot issues on switches and access ports.

Procedure

Step 1 Display the statistics gathered by the NBAR Protocol Discovery feature.

Example:

```
Device# show ip nbar protocol-discovery
```

Optionally, use additional keywords to fine-tune the statistics displayed:

- **interface** *interface-type interface-number*: specifies a particular interface

- **stats** {**byte-count** | **bit-rate** | **packet-count** | **max-bit-rate**}: displays specific statistics types
- **protocol** *protocol-name*: filters by protocol name
- **top-n** *number*: displays top n protocols

Step 2 Display information about policy map applied to the interface.

Example:

```
Device# show policy-map interface interface-type interface-number
```

Step 3 Display statistics about all flows on the specified switch.

Example:

```
Device# show platform software fed switch switch-id wdavc flows
```

Verify Application Visibility and Control

Use this procedure to verify the configuration and functionality of application visibility and control in a wired network.

Procedure

Step 1 Verify the configuration.

a) Display a report of the Protocol Discovery statistics per interface.

Example:

```
Device# show ip nbar protocol-discovery int GigabitEthernet1/0/1


GigabitEthernet1/0/1
Last clearing of "show ip nbar protocol-discovery" counters 00:03:16

                                    Input                                     Output
                                    -----                                     ------
Protocol                            Packet Count                              Packet Count

                                    Byte Count                                Byte Count
                                    30sec Bit Rate (bps)                      30sec Bit Rate
(bps)
                                   30sec Max Bit Rate (bps)                  30sec Max Bit Rate
 (bps)
------------------------ ------------------------
-----------------------------------------------------
ms-lync                             60580                                     55911
                                    31174777                                  28774864
                                    3613000                                   93000
                                    3613000                                   3437000
Total                               60580                                     55911
```

```
                                        31174777                        28774864
                                        3613000                         93000
                                        3613000                         3437000
```

b) Display the QoS statistics and the configured policy maps on all interfaces.

Example:

```
Device# show policy-map int


GigabitEthernet1/0/1
 Service-policy input: MARKING-IN

   Class-map: NBAR-VOICE (match-any)
     718 packets
     Match: protocol ms-lync-audio
       0 packets, 0 bytes
       30 second rate 0 bps
     QoS Set
       dscp ef

   Class-map: NBAR-MM_CONFERENCING (match-any)
     6451 packets
     Match: protocol ms-lync
       0 packets, 0 bytes
       30 second rate 0 bps
     Match: protocol ms-lync-video
       0 packets, 0 bytes
       30 second rate 0 bps
     QoS Set
       dscp af41

   Class-map: class-default (match-any)
     34 packets
     Match: any
```

Step 2 Verify attributes-based QoS configuration.

a) Display the attribute-based QoS statistics and the configured policy maps on all interfaces.

Example:

```
Device# show policy-map interface gigabitEthernet 1/0/2
 GigabitEthernet1/0/2

  Service-policy input: attrib--rel-types

    Class-map: rel-relevant (match-all)
      20 packets
      Match: protocol attribute business-relevance business-relevant
      QoS Set
        dscp ef

    Class-map: rel-irrelevant (match-all)
      0 packets
      Match: protocol attribute business-relevance business-irrelevant
```

```
    QoS Set
      dscp af11

  Class-map: rel-default (match-all)
    14 packets
    Match: protocol attribute business-relevance default
    QoS Set
      dscp default

  Class-map: class-default (match-any)
    0 packets
    Match: any
```

b) Display all the protocol attributes used by NBAR.

Example:

```
Device# show ip nbar protocol-attribute cisco-jabber-im
           Protocol Name : cisco-jabber-im
                encrypted : encrypted-yes
                   tunnel : tunnel-no
                 category : voice-and-video
             sub-category : enterprise-media-conferencing
        application-group : cisco-jabber-group
           p2p-technology : p2p-tech-no
            traffic-class : transactional-data
       business-relevance : business-relevant
          application-set : collaboration-apps

Device# show ip nbar protocol-attribute google-services
            Protocol Name : google-services
                encrypted : encrypted-yes
                   tunnel : tunnel-no
                 category : other
              sub-category : other
         application-group : google-group
           p2p-technology : p2p-tech-yes
            traffic-class : transactional-data
       business-relevance : default
          application-set : general-browsing

Device# show ip nbar protocol-attribute dns
            Protocol Name : google-services
                encrypted : encrypted-yes
                   tunnel : tunnel-no
                 category : other
              sub-category : other
         application-group : google-group
           p2p-technology : p2p-tech-yes
            traffic-class : transactional-data
       business-relevance : default
          application-set : general-browsing
```

```
Device# show ip nbar protocol-attribute unknown
            Protocol Name : unknown
                encrypted : encrypted-no
                   tunnel : tunnel-no
                 category : other
             sub-category : other
        application-group : other
           p2p-technology : p2p-tech-no
            traffic-class : bulk-data
       business-relevance : default
          application-set : general-misc
```

Step 3 Verify flow monitor configuration.

a) Display information about the specified wired AVC flow monitor.

Example:

```
Device # show flow monitor wdavc

Flow Monitor wdavc:
  Description:       User defined
  Flow Record:       wdavc
  Flow Exporter:     wdavc-exp (inactive)
  Cache:
    Type:                normal (Platform cache)
    Status:              not allocated
    Size:                12000 entries
    Inactive Timeout:    15 secs
    Active Timeout:      1800 secs
```

b) Display statistics for wired AVC flow monitor.

Example:

```
Device# show flow monitor wdavc statistics
  Cache type:                                 Normal (Platform cache)
  Cache size:                                  12000
  Current entries:                                13

  Flows added:                                    26
  Flows aged:                                     13
    - Active timeout      (  1800 secs)           1
    - Inactive timeout    (    15 secs)          12
```

You can clear the statistics of the specified flow monitor. Use the **show flow monitor wdavc statistics** command after using the **clear flow monitor wdavc statistics** command to verify that all the statistics have been reset.

This is a sample output after clearing flow monitor statistics:

```
Device# show flow monitor wdavc statistics
  Cache type:                               Normal (Platform cache)
  Cache size:                                12000
  Current entries:                               0

  Flows added:                                   0
  Flows aged:                                    0
```

Step 4 View cache content.

- Display flow cache contents in a tabular format.

```
Device# show flow monitor wdavc cache format table
  Cache type:                                Normal (Platform cache)
  Cache size:                                 12000
  Current entries:                               13

  Flows added:                                   26
  Flows aged:                                    13
    - Active timeout      (  1800 secs)           1
    - Inactive timeout    (    15 secs)          12

CONN IPV4 INITIATOR ADDR  CONN IPV4 RESPONDER ADDR  CONN RESPONDER
PORT  FLOW OBSPOINT ID  IP VERSION  IP PROT  APP NAME
     flow dirn ...................
------------------------  ------------------------  -------------------
  ----------------  ----------  -------  --------------------------
 ---------
64.103.125.147            144.254.71.184
53        4294967305           4       17  port dns
   Input     .....................
64.103.121.103            10.1.1.2
67        4294967305           4       17  layer7 dhcp
   Input     ....contd...........
64.103.125.3              64.103.125.97
68        4294967305           4       17  layer7 dhcp
   Input     .....................
10.0.2.6                  157.55.40.149
443        4294967305           4        6  layer7 ms-lync
    Input     .....................
64.103.126.28             66.163.36.139
443        4294967305           4        6  layer7 cisco-jabber-im
    Input     ....contd...........
64.103.125.2              64.103.125.29
68        4294967305           4       17  layer7 dhcp
   Input     .....................
64.103.125.97             64.103.101.181
67        4294967305           4       17  layer7 dhcp
   Input     .....................
192.168.100.6             10.10.20.1
5060       4294967305          4      17  layer7 cisco-jabber-control
  Input     ....contd...........
64.103.125.3              64.103.125.29
68        4294967305           4       17  layer7 dhcp
   Input     .....................
10.80.101.18              10.80.101.6
5060       4294967305          4       6  layer7 cisco-collab-control
  Input     .....................
10.1.11.4                 66.102.11.99
80        4294967305           4        6  layer7 google-services
   Input     ....contd...........
```

```
64.103.125.2              64.103.125.97
68        4294967305           4       17  layer7 dhcp
   Input     .....................
64.103.125.29             64.103.101.181
67        4294967305           4       17  layer7 dhcp
   Input     .....................
```

- Display flow cache contents in similar format as the flow record.

```
Device# show flow monitor wdavc cache format record
  Cache type:                                 Normal (Platform cache)
  Cache size:                                  12000
  Current entries:                                13

  Flows added:                                    26
  Flows aged:                                     13
    - Active timeout      (  1800 secs)            1
    - Inactive timeout    (    15 secs)           12

CONNECTION IPV4 INITIATOR ADDRESS:        64.103.125.147
CONNECTION IPV4 RESPONDER ADDRESS:        144.254.71.184
CONNECTION RESPONDER PORT:                53
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         port dns
flow direction:                           Input
timestamp abs first:                      08:55:46.917
timestamp abs last:                       08:55:46.917
connection initiator:                     Initiator
connection count new:                     2
connection server packets counter:        1
connection client packets counter:        1
connection server network bytes counter:  190
connection client network bytes counter:  106

CONNECTION IPV4 INITIATOR ADDRESS:        64.103.121.103
CONNECTION IPV4 RESPONDER ADDRESS:        10.1.1.2
CONNECTION RESPONDER PORT:                67
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 dhcp
flow direction:                           Input
timestamp abs first:                      08:55:47.917
timestamp abs last:                       08:55:47.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        1
connection server network bytes counter:  0
connection client network bytes counter:  350
```

```
CONNECTION IPV4 INITIATOR ADDRESS:        64.103.125.3
CONNECTION IPV4 RESPONDER ADDRESS:        64.103.125.97
CONNECTION RESPONDER PORT:                68
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 dhcp
flow direction:                           Input
timestamp abs first:                      08:55:47.917
timestamp abs last:                       08:55:53.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        4
connection server network bytes counter:  0
connection client network bytes counter:  1412

CONNECTION IPV4 INITIATOR ADDRESS:        10.0.2.6
CONNECTION IPV4 RESPONDER ADDRESS:        157.55.40.149
CONNECTION RESPONDER PORT:                443
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              6
APPLICATION NAME:                         layer7 ms-lync
flow direction:                           Input
timestamp abs first:                      08:55:46.917
timestamp abs last:                       08:55:46.917
connection initiator:                     Initiator
connection count new:                     2
connection server packets counter:        10
connection client packets counter:        14
connection server network bytes counter:  6490
connection client network bytes counter:  1639

CONNECTION IPV4 INITIATOR ADDRESS:        64.103.126.28
CONNECTION IPV4 RESPONDER ADDRESS:        66.163.36.139
CONNECTION RESPONDER PORT:                443
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              6
APPLICATION NAME:                         layer7 cisco-jabber-im
flow direction:                           Input
timestamp abs first:                      08:55:46.917
timestamp abs last:                       08:55:46.917
connection initiator:                     Initiator
connection count new:                     2
connection server packets counter:        12
connection client packets counter:        10
connection server network bytes counter:  5871
connection client network bytes counter:  2088
```

```
CONNECTION IPV4 INITIATOR ADDRESS:        64.103.125.2
CONNECTION IPV4 RESPONDER ADDRESS:        64.103.125.29
CONNECTION RESPONDER PORT:                68
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 dhcp
flow direction:                           Input
timestamp abs first:                      08:55:47.917
timestamp abs last:                       08:55:47.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        2
connection server network bytes counter:  0
connection client network bytes counter:  712

CONNECTION IPV4 INITIATOR ADDRESS:        64.103.125.97
CONNECTION IPV4 RESPONDER ADDRESS:        64.103.101.181
CONNECTION RESPONDER PORT:                67
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 dhcp
flow direction:                           Input
timestamp abs first:                      08:55:47.917
timestamp abs last:                       08:55:47.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        1
connection server network bytes counter:  0
connection client network bytes counter:  350

CONNECTION IPV4 INITIATOR ADDRESS:        192.168.100.6
CONNECTION IPV4 RESPONDER ADDRESS:        10.10.20.1
CONNECTION RESPONDER PORT:                5060
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 cisco-jabber-control
flow direction:                           Input
timestamp abs first:                      08:55:46.917
timestamp abs last:                       08:55:46.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        2
connection server network bytes counter:  0
connection client network bytes counter:  2046
```

```
CONNECTION IPV4 INITIATOR ADDRESS:        64.103.125.3
CONNECTION IPV4 RESPONDER ADDRESS:        64.103.125.29
CONNECTION RESPONDER PORT:                68
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 dhcp
flow direction:                           Input
timestamp abs first:                      08:55:47.917
timestamp abs last:                       08:55:47.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        2
connection server network bytes counter:  0
connection client network bytes counter:  712

CONNECTION IPV4 INITIATOR ADDRESS:        10.80.101.18
CONNECTION IPV4 RESPONDER ADDRESS:        10.80.101.6
CONNECTION RESPONDER PORT:                5060
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              6
APPLICATION NAME:                         layer7 cisco-collab-control
flow direction:                           Input
timestamp abs first:                      08:55:46.917
timestamp abs last:                       08:55:47.917
connection initiator:                     Initiator
connection count new:                     2
connection server packets counter:        23
connection client packets counter:        27
connection server network bytes counter:  12752
connection client network bytes counter:  8773

CONNECTION IPV4 INITIATOR ADDRESS:        10.1.11.4
CONNECTION IPV4 RESPONDER ADDRESS:        66.102.11.99
CONNECTION RESPONDER PORT:                80
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              6
APPLICATION NAME:                         layer7 google-services
flow direction:                           Input
timestamp abs first:                      08:55:46.917
timestamp abs last:                       08:55:46.917
connection initiator:                     Initiator
connection count new:                     2
connection server packets counter:        3
connection client packets counter:        5
connection server network bytes counter:  1733
connection client network bytes counter:  663
```

```
CONNECTION IPV4 INITIATOR ADDRESS:        64.103.125.2
CONNECTION IPV4 RESPONDER ADDRESS:        64.103.125.97
CONNECTION RESPONDER PORT:                68
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 dhcp
flow direction:                           Input
timestamp abs first:                      08:55:47.917
timestamp abs last:                       08:55:53.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        4
connection server network bytes counter:  0
connection client network bytes counter:  1412

CONNECTION IPV4 INITIATOR ADDRESS:        64.103.125.29
CONNECTION IPV4 RESPONDER ADDRESS:        64.103.101.181
CONNECTION RESPONDER PORT:                67
FLOW OBSPOINT ID:                         4294967305
IP VERSION:                               4
IP PROTOCOL:                              17
APPLICATION NAME:                         layer7 dhcp
flow direction:                           Input
timestamp abs first:                      08:55:47.917
timestamp abs last:                       08:55:47.917
connection initiator:                     Initiator
connection count new:                     1
connection server packets counter:        0
connection client packets counter:        1
connection server network bytes counter:  0
connection client network bytes counter:  350
```

- Display flow cache contents in CSV format.

```
Device# show flow monitor wdavc cache format csv
  Cache type:                                 Normal (Platform cache)
  Cache size:                                  12000
  Current entries:                                13

  Flows added:                                    26
  Flows aged:                                     13
    - Active timeout      (  1800 secs)           1
    - Inactive timeout    (    15 secs)          12

CONN IPV4 INITIATOR ADDR,CONN IPV4 RESPONDER ADDR,CONN RESPONDER
PORT,FLOW OBSPOINT ID,IP VERSION,IP
PROT,APP NAME,flow dirn,time abs first,time abs last,conn initiator,conn
 count new,conn server packets
cnt,conn client packets cnt,conn server network bytes cnt,conn client
```

```
 network bytes cnt
64.103.125.147,144.254.71.184,53,4294967305,4,17,port
dns,Input,08:55:46.917,08:55:46.917,Initiator,2,1,1,190,106
64.103.121.103,10.1.1.2,67,4294967305,4,17,layer7
dhcp,Input,08:55:47.917,08:55:47.917,Initiator,1,0,1,0,350
64.103.125.3,64.103.125.97,68,4294967305,4,17,layer7
dhcp,Input,08:55:47.917,08:55:53.917,Initiator,1,0,4,0,1412
10.0.2.6,157.55.40.149,443,4294967305,4,6,layer7 ms-
lync,Input,08:55:46.917,08:55:46.917,Initiator,2,10,14,6490,1639
64.103.126.28,66.163.36.139,443,4294967305,4,6,layer7 cisco-jabber-
im,Input,08:55:46.917,08:55:46.917,Initiator,2,12,10,5871,2088
64.103.125.2,64.103.125.29,68,4294967305,4,17,layer7
dhcp,Input,08:55:47.917,08:55:47.917,Initiator,1,0,2,0,712
64.103.125.97,64.103.101.181,67,4294967305,4,17,layer7
dhcp,Input,08:55:47.917,08:55:47.917,Initiator,1,0,1,0,350
192.168.100.6,10.10.20.1,5060,4294967305,4,17,layer7 cisco-jabber-
control,Input,08:55:46.917,08:55:46.917,Initiator,1,0,2,0,2046
64.103.125.3,64.103.125.29,68,4294967305,4,17,layer7
dhcp,Input,08:55:47.917,08:55:47.917,Initiator,1,0,2,0,712
10.80.101.18,10.80.101.6,5060,4294967305,4,6,layer7 cisco-collab-
control,Input,08:55:46.917,08:55:47.917,Initiator,2,23,27,12752,8773
10.1.11.4,66.102.11.99,80,4294967305,4,6,layer7 google-
services,Input,08:55:46.917,08:55:46.917,Initiator,2,3,5,1733,663
64.103.125.2,64.103.125.97,68,4294967305,4,17,layer7
dhcp,Input,08:55:47.917,08:55:53.917,Initiator,1,0,4,0,1412
64.103.125.29,64.103.101.181,67,4294967305,4,17,layer7
dhcp,Input,08:55:47.917,08:55:47.917,Initiator,1,0,1,0,350
```

Troubleshooting guidelines for Application Visibility and Control

Follow these guidelines to resolve common issues and ensure optimal performance when configuring wired Application Visibility and Control:

- Avoid Control and Provisioning of Wireless Access Points (CAPWAP) classification issues: Ensure that NBAR is enabled only on access ports that are not connected to wireless access points. Traffic originating from access points is classified as CAPWAP, as actual classification occurs at the AP or WLC level.
- Prevent asymmetric traffic visibility: Attach NBAR only to access ports where both sides of the traffic flow are visible. Asymmetric traffic—where one side is classified on a different switch member—results in incomplete classification and unknown traffic reports. Do not attach NBAR to uplinks or interfaces that are part of a port channel.
- View traffic distribution over time: Use the WebUI to access a historical view of traffic distribution for the last 48 hours.
- Manage egress policy limitations: The queue-based egress policies are not supported with **match protocol** commands. Use **set DSCP** on ingress and perform shaping on egress based on the DSCP value.

- Understand global NBAR activation: NBAR is globally activated on the stack if any class map uses the **match protocol** command. This is expected behavior and does not consume system resources if no traffic is subjected to classification.
- Account for initial flow classification latency: Expect some traffic to appear in the default QoS queue for new flows. It takes a few packets to classify a flow and install the result in the hardware; during this interval, the classification remains unknown.