

## Chapter 36

# CoS Configuration Guidelines

To configure class-of-service (CoS) properties, you can include the following statements at the [edit class-of-service] hierarchy level of the configuration:

```
class-of-service {
  classifiers {
    type classifier-name {
      import (classifier-name | default);
      forwarding-class class-name {
        loss-priority level {
          code-points [ aliases ] [ 6-bit-patterns ];
        }
      }
    }
  }
  code-point-aliases {
    (dscp | dscp-ipv6 | exp | ieee-802.1 | inet-precedence) {
      alias-name bits;
    }
  }
  drop-profiles {
    profile-name {
      fill-level percentage drop-probability percentage;
      interpolate {
        drop-probability [ values ];
        fill-level [ values ];
      }
    }
  }
  fabric {
    scheduler-map {
      priority (high | low) scheduler scheduler-name;
    }
  }
  forwarding-classes {
    queue queue-number class-name priority (high | low);
  }
  forwarding-policy {
    next-hop-map map-name {
      forwarding-class class-name {
        next-hop [ next-hop-name ];
        lsp-next-hop [ lsp-regular-expression ];
      }
    }
  }
}
```

```

class class-name {
  classification-override {
    forwarding-class class-name;
  }
}
}
interfaces
  interface-name {
    scheduler-map map-name;
    scheduler-map-chassis map-name;
    unit logical-unit-number {
      classifiers {
        (dscp | dscp-ipv6 | exp | ieee-802.1 | inet-precedence) (classifier-name
          | default);
      }
      forwarding-class class-name;
      rewrite-rules {
        dscp (rewrite-name | default);
        dscp-ipv6 (rewrite-name | default);
        exp (rewrite-name | default) protocol protocol-types;
        exp-push-push-push default;
        exp-swap-push-push default;
        ieee-802.1 (rewrite-name | default);
        inet-precedence (rewrite-name | default);
      }
      scheduler-map map-name;
      shaping-rate rate;
    }
  }
}
restricted-queues {
  forwarding-class class-name queue queue-number;
}
rewrite-rules {
  (dscp | dscp-ipv6 | exp | ieee-802.1 | inet-precedence) rewrite-name {
    import (rewrite-name | default);
    forwarding-class class-name {
      loss-priority level code-point (alias | bits);
    }
  }
}
scheduler-maps {
  map-name {
    forwarding-class class-name scheduler scheduler-name;
  }
}
schedulers {
  scheduler-name {
    buffer-size (percent percentage | remainder | temporal microseconds);
    drop-profile-map loss-priority (any | high | low) protocol (any | non-tcp | tcp)
      drop-profile profile-name;
    priority priority-level;
    transmit-rate (rate | percent percentage | remainder) <exact>;
  }
}
}

```

The following RFCs define the standards supported by certain aspects of the CoS software:

*RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*

*RFC 2597, Assured Forwarding PHB Group*

*RFC 2598, An Expedited Forwarding PHB*

*RFC 2983, Diffserv and Tunnels*, is not supported.

The JUNOS software supports only two loss priorities and, by default, supports only one assured forwarding (AF) class, although you can configure more at the expense of other class types.

This chapter includes the following sections:

Hardware Capabilities and Limitations on page 816

Defining Code-Point Aliases on page 818

Configuring Forwarding Classes on page 821

Classifying Packets by Behavior Aggregate on page 830

Configuring Scheduling Maps on page 835

Configuring RED Drop Profiles on page 853

Rewriting Packet Header Information on page 854

Configuring CoS-Based Forwarding on page 863

Configuring Class of Service for Tunnels on page 869

Examples: Configuring Class of Service on page 873

## Hardware Capabilities and Limitations

Juniper Networks T-series platforms and M-series platforms with an enhanced FPC have more CoS capabilities than M-series platforms that use the earlier FPC model. Table 47 lists the differences between the FPC and the enhanced FPC.

**Table 47: CoS Hardware Capabilities and Limitations**

Feature	M-series FPC	M-series Enhanced FPC	M320 and T-series FPC	Comments
<b>Classifiers</b>				
Maximum number per FPC or PIC	1	8	64	For M-series FPCs, the one-classifier limit includes the default IP precedence classifier. If you create a new classifier and apply it to an interface, the new classifier does not override the default classifier for other interfaces on the same FPC. The general rule for classifier replacement is that the first classifier associated with a logical interface is the one that is used. The default classifier can be replaced only when a single interface is associated with the default classifier. For more information, see Table 51 on page 833.
dscp	No	Yes	Yes	On all platforms, you cannot configure IP precedence and DSCP classifiers on a single logical interface, because both apply to IPv4 packets. For more information, see Table 51 on page 833.
dscp-ipv6	No	Yes	Yes	For T-series platforms, you can apply separate classifiers for IPv4 and IPv6 packets per logical interface. For M-series enhanced FPCs, you cannot apply separate classifiers for IPv4 and IPv6 packets, and classifier assignment works as follows: If you assign a DSCP classifier only, IPv4 and IPv6 packets are classified using the DSCP classifier. If you assign an IP precedence classifier only, IPv4 and IPv6 packets are classified using the IP precedence classifier. In this case, the lower three bits of the DSCP field are ignored because IP precedence mapping requires the upper three bits only. If you assign either the DSCP or the IP precedence classifier in conjunction with the DSCP IPv6 classifier, the commit fails. If you assign a DSCP IPv6 classifier only, IPv4 and IPv6 packets are classified using the DSCP IPv6 classifier, but the commit displays a warning message. For more information, see Table 51 on page 833.
ieee-802.1p	No	Yes	Yes	On M-series FPC and T-series platforms, if you associate an IEEE 802.1p classifier with a logical interface, you cannot associate any other classifier with that logical interface. For more information, see Table 51 on page 833.
inet-precedence	Yes	Yes	Yes	On all platforms, you cannot assign IP precedence and DSCP classifiers to a single logical interface, because both apply to IPv4 packets. For more information, see Table 51 on page 833.
mpls-exp	Yes	Yes	Yes	For M-series FPCs, only the default MPLS EXP classifier is supported; the default MPLS EXP classifier takes the EXP bits 1 and 2 as the output queue number.

Feature	M-series FPC	M-series Enhanced FPC	M320 and T-series FPC	Comments
<b>Rewrite Markers</b>				
Maximum number per FPC or PIC	0	0	64	
dscp	No	Yes	Yes	For M320 and T-series FPCs, you must decode the loss priority using the firewall filter before you can use loss priority to select the rewrite code point. For more information, see "Overriding the Default PLP on M320 and T-series Platforms" on page 834.
dscp-ipv6	No	Yes	Yes	For M320 and T-series FPCs, you must decode the loss priority using the firewall filter before you can use loss priority to select the rewrite code point. For more information, see "Overriding the Default PLP on M320 and T-series Platforms" on page 834.
ieee-802.1	No	Yes	Yes	For M-series Enhanced FPCs and T-series FPCs, fixed rewrite loss priority determines the value for bit 0; queue number (forwarding class) determines bits 1 and 2.
inet-precedence	Yes	Yes	Yes	For M320 and T-series FPCs, you must decode the loss priority using the firewall filter before you can use loss priority to select the rewrite code point. For more information, see "Overriding the Default PLP on M320 and T-series Platforms" on page 834.
mpls-exp	Yes	Yes	Yes	For M320 and T-series FPCs, you must decode the loss priority using the firewall filter before you can use loss priority to select the rewrite code point. For more information, see "Overriding the Default PLP on M320 and T-series Platforms" on page 834.  For M-series FPCs, fixed rewrite loss priority determines the value for bit 0; queue number (forwarding class) determines bits 1 and 2.
<b>Queuing</b>				
Priority	No	Yes	Yes	The medium-low and medium-high priorities are supported differently, depending on FPC type. For more information, see Table 52 on page 840.
<b>Drop Profiles</b>				
Maximum number per FPC or PIC	2	16	32	
Per queue	No	Yes	Yes	
Per loss priority	Yes	Yes	Yes	
Per TCP bit	No	Yes	Yes	

## Defining Code-Point Aliases

A code-point alias is a name you assign to a set of DiffServ code point (DSCP) and DSCP IPv6 bits. When you configure classes and define classifiers, you can refer to the code points by these alias names. You can configure user-defined classifiers in terms of alias names. If the value of an alias changes, it alters the behavior of any classifier that references that alias.

Table 48 shows the default mappings between the bit values and standard aliases. For example, it is widely accepted that the alias for DSCP 101110 is ef (expedited forwarding).

**Table 48: Default DSCP Mappings**

DiffServ Code Designator	Mapping
<b>DSCP and DSCP IPv6 Code Points</b>	
ef	101110
af11	001010
af12	001100
af13	001110
af21	010010
af22	010100
af23	010110
af31	011010
af32	011100
af33	011110
af41	100010
af42	100100
af43	100110
be	000000
cs1	001000
cs2	010000
cs3	011000
cs4	100000
cs5	101000
nc1/cs6	110000
nc2/cs7	111000

<b>DiffServ Code Designator</b>	<b>Mapping</b>
<b>MPLS EXP Code Points</b>	
be	000
be1	001
ef	010
ef1	011
af11	100
af12	101
nc1/cs6	110
nc2/cs7	111
<b>IEEE 802.1 Code Points</b>	
be	000
be1	001
ef	010
ef1	011
af11	100
af12	101
nc1/cs6	110
nc2/cs7	111
<b>Legacy IP Precedence Code Points</b>	
be	000
be1	001
ef	010
ef1	011
af11	100
af12	101
nc1/cs6	110
nc2/cs7	111

You use code-point aliases to do the following:

- Define an alias for bits that currently have no alias

- Define multiple aliases for the same bits

- Redefine an alias name to mean a different set of bits than the default

To define a code-point alias, include the `code-point-aliases` statement at the `[edit class-of-service]` hierarchy level:

```
[edit class-of-service]
code-point-aliases {
  (dscp | dscp-ipv6 | exp | ieee-802.1 | inet-precedence) {
    alias-name bits;
  }
}
```

For example, you might set up the following configuration:

```
[edit class-of-service]
code-point-aliases {
  dscp {
    my1 110001;
    my2 101110;
    be 000001;
    cs7 110000;
  }
}
```

The sample configuration produces this mapping:

```
user@host>show class-of-service code-point-aliases dscp
Code point type: dscp
Alias      Bit pattern
ef/my2    101110
af11      001010
af12      001100
af13      001110
af21      010010
af22      010100
af23      010110
af31      011010
af32      011100
af33      011110
af41      100010
af42      100100
af43      100110
be        000001
cs1       001000
cs2       010000
cs3       011000
cs4       100000
cs5       101000
nc1/cs6/cs7 110000
nc2       111000
my1       110001
```

The following notes explain certain results in the mapping:

my1 110001:

110001 was not mapped to anything before, and my1 is a new alias.

Nothing in the default mapping table is changed by this statement.

my2 101110:

101110 is now mapped to my2 as well as ef.

be 000001:

be is now mapped to 000001.

The old value of be, 000000, is not associated with any alias. Packets with this DSCP value are now classified to the default forwarding class.

cs7 110000:

cs7 is now mapped to 110000, as well as nc1 and cs6.

The old value of cs7, 111000, is still mapped to nc2.

## Configuring Forwarding Classes

Forwarding classes replace output queues from the previous CoS configuration command set. You assign each forwarding class to an internal queue number by including the forwarding-classes statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
forwarding-classes {
  queue queue-number class-name;
}
}
```



**NOTE:** You cannot commit a configuration that assigns the same forwarding class to two different queues.

Table 49 shows the four forwarding classes defined by default.

**Table 49: Default Forwarding Classes**

Forwarding Class Name	Queue
best-effort	queue 0
expedited-forwarding	queue 1
assured-forwarding	queue 2
network-control	queue 3

The following rules govern queue assignment:

If classifiers fail to classify a packet, the packet always receives the default classification to the class associated with queue 0.

The number of queues is dependent on the hardware plugged into the chassis. CoS configurations are inherently contingent on the number of queues on the system. Only two classes, best-effort and network-control, are actually referenced in the default configuration. The default configuration works on any platform.

CoS configurations that specify more queues than the platform can support are not accepted. The commit fails with a detailed message that states the total number of queues available.

All default CoS configuration is based on queue number. The name of the forwarding class that shows up when the default configuration is displayed is the forwarding class currently associated with that queue.

This is the default configuration for forwarding-classes:

```
[edit class-of-service]
forwarding-classes {
  queue 0 best-effort;
  queue 1 expedited-forwarding;
  queue 2 assured-forwarding;
  queue 3 network-control;
}
```

If you reassign the forwarding-class names, the best-effort forwarding-class name appears in the locations in the configuration previously occupied by network-control as follows:

```
forwarding-classes {
  queue 0 network-control;
  queue 1 assured-forwarding;
  queue 2 expedited-forwarding;
  queue 3 best-effort;
}
```

All the default rules of classification and scheduling that applied to queue 3 still apply. Queue 3 is simply now renamed best-effort.

In the current default configuration:

Only IP precedence classifiers are associated with interfaces.

The only classes designated are best-effort and network-control.

Schedulers are not defined for the expedited-forwarding or assured-forwarding classes.

You must make a conscious effort to classify packets to the expedited-forwarding or assured-forwarding class and define schedulers for these classes.

For ATM interfaces on M-series platforms, when you use fixed classification with multiple logical interfaces classifying to separate queues, a logical interface without a classifier attached inherits the most recent classifier applied on a different logical interface. For example, suppose you configure traffic through logical unit 0 to be classified into Q1, and you configure traffic through logical unit 1 to be classified into Q3. You want traffic through logical unit 2 to be classified into the default classifier, which is Q0. In this case, traffic through logical unit 2 is classified into Q3, because the configuration of logical unit 1 was committed last.

For more information about configuring forwarding classes on M320 and T-series platforms, see the following special topics:

Overriding Fabric Priority Queuing on page 823

Configuring up to Eight Forwarding Classes on page 824

### **Assigning a Forwarding Class to an Interface**

To assign the forwarding class configuration to the output logical interface, include the forwarding-class statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy level:

```
[edit class-of-service interfaces interface-name unit logical-unit-number]
forwarding-class class-name;
```

You can include interface wildcards for *interface-name* and *logical-unit-number*.

### **Overriding Fabric Priority Queuing**

For M320 and T-series platforms only, you can override automatic fabric priority queuing. For egress interfaces, fabric priority queuing matches the queue priority you assign at the [edit class-of-service schedulers *scheduler-name*] hierarchy level. High-priority egress traffic is automatically assigned to high-priority fabric queues. Likewise, low-priority egress traffic is automatically assigned to low-priority fabric queues.

You can override the default fabric priority queuing of egress traffic by including the priority statement at the [edit class-of-service forwarding-classes queue *queue-number* *class-name*] hierarchy level:

```
[edit class-of-service forwarding-classes queue queue-number class-name]
priority (high | low);
```

For information about associating a scheduler with a fabric priority, see “Associating a Scheduler with a Fabric Priority” on page 851.

## Configuring up to Eight Forwarding Classes

By default on M-series and T-series platforms, four queues are mapped to four forwarding classes, as shown in Table 49 on page 821. For M320 and T-series platforms only, you can configure more than four forwarding classes and queues. To do this, include the queue statement at the [edit class-of-service forwarding-classes] hierarchy level:

```
[edit class-of-service forwarding-classes]
queue queue-number class-name;
```

For all M320 and T-series PICs, you can configure up to eight forwarding classes. For most T-series PICs, the output queue number can be from 0 through 7, and you must map the forwarding classes one-to-one with the output queues. This is the global configuration.

By default, the scheduler transmission rate and buffer size percentages for queues 0 through 7 are 95, 0, 0, 5, 0, 0, 0, and 0 percent.

For more detail, see the following sections:

PICs Restricted to Four Queues on page 824

Enabling Eight Queues on IQ Interfaces on page 825

Examples: Configuring up to Eight Forwarding Classes on page 826

### PICs Restricted to Four Queues

Some T-series PICs support up to eight forwarding classes and are restricted to four queues. For these PICs only, you can assign multiple forwarding classes to single queues. Contact Juniper Networks customer support for a current list of T-series PICs that are restricted to four queues. To determine how many queues an interface supports, you can check the CoS queues output field of the show interfaces *interface-name* extensive command:

```
user@host> show interfaces so-1/0/0 extensive
CoS queues: 8 supported
```

By default, for T-series PICs that are restricted to four queues, the routing platform overrides the global configuration based on the following formula:

$$Q_r = Q_d \text{ mod } R_{\text{max}}$$

$Q_r$  is the queue number assigned if the PIC is restricted to four queues.

$Q_d$  is the queue number that would have been mapped if this PIC were not restricted.

$R_{\text{max}}$  is the maximum number of restricted queues available. Currently, this is four.

For example, assume you map the forwarding class ef to queue 6. For a PIC restricted to four queues, the queue number for forwarding class ef is  $Q_r = 6 \text{ mod } 4 = 2$ .

To determine which queue is assigned to a forwarding class, use the `show class-of-service forwarding-class` command from the top level of the CLI. The output shows queue assignments for both global queue mappings and restricted queue mappings:

```
user@host> show class-of-service forwarding-class
Forwarding class      Queue  Restricted Queue  Fabric priority
be                    0      2                 low
ef                    1      2                 low
assured-forwarding   2      2                 low
network-control       3      3                 low
```

For T-series PICs restricted to four queues, you can override the formula-derived queue assignment by including the `restricted-queues` statement at the `[edit class-of-service]` hierarchy level:

```
[edit class-of-service]
restricted-queues {
    forwarding-class class-name queue queue-number;
}
```

You can configure up to eight forwarding classes. The output queue number can be from 0 through 3. Therefore, for PICs restricted to four queues, you can map multiple forwarding classes to single queues. If you map multiple forwarding classes to a queue, the multiple forwarding classes must refer to the same scheduler. The class name you configure at the `[edit class-of-service restricted-queues]` hierarchy level must be either a default forwarding class name from Table 49 on page 821 or a forwarding class you configure at the `[edit class-of-service forwarding-classes]` hierarchy level.

### Enabling Eight Queues on IQ Interfaces

By default, Intelligent Queuing (IQ) PICs on T-series and M320 platforms are restricted to a maximum of four egress queues per interface. You can enable eight egress queues on IQ interfaces by including the `max-queues-per-interface` statement at the `[edit chassis fpc slot-number pic pic-number]` hierarchy level:

```
[edit chassis fpc slot-number pic pic-number]
max-queues-per-interface number;
```

The numerical value can be 4 or 8.

If you include the `max-queues-per-interface` statement, all ports on the IQ PIC use the configured mode.

When you change modes between four queues and eight queues, all physical interfaces on the PIC are deleted and readded.

For more information, see the *JUNOS System Basics Configuration Guide*.

### Examples: Configuring up to Eight Forwarding Classes

Configure a one-to-one mapping between eight forwarding classes and eight queues.

For PICs restricted to four queues, map two forwarding classes to each queue.

```
[edit class-of-service]
forwarding-classes {
  queue 0 be;
  queue 1 ef;
  queue 2 af;
  queue 3 nc;
  queue 4 ef1;
  queue 5 ef2;
  queue 6 af1;
  queue 7 nc1;
}
```

#### Mapping Two Forwarding Classes to Each Queue

```
[edit class-of-service]
restricted-queues {
  forwarding-class be queue 0;
  forwarding-class be1 queue 0;
  forwarding-class ef queue 1;
  forwarding-class ef1 queue 1;
  forwarding-class af queue 2;
  forwarding-class af1 queue 2;
  forwarding-class nc queue 3;
  forwarding-class nc1 queue 3;
}
```

#### Defining Eight Classifiers

```
[edit class-of-service]
classifiers {
  dscp dscp-table {
    forwarding-class ef {
      loss-priority low code-points [101000, 101001];
      loss-priority high code-points [101010, 101011];
    }
    forwarding-class af {
      loss-priority low code-points [010000, 010001];
      loss-priority high code-points [010010, 010011];
    }
    forwarding-class be {
      loss-priority low code-points [000000];
    }
    forwarding-class nc {
      loss-priority low code-points [111000];
    }
    forwarding-class ef1 {
      loss-priority low code-points [101100, 101101];
      loss-priority high code-points [101110];
    }
    forwarding-class af1 {
      loss-priority high code-points [101110];
    }
  }
}
```

```

        forwarding-class ef2 {
            loss-priority low code-points [101111];
        }
        forwarding-class af2 {
            loss-priority low code-points [010000];
        }
    }
}

```

### Adding Eight Schedulers to a Scheduler Map

Configure a custom scheduler map that applies globally to all interfaces, except those that are restricted to four queues.

```

[edit class-of-service]
scheduler-maps {
    interface-global {
        forwarding-class be scheduler Q0;
        forwarding-class ef scheduler Q1;
        forwarding-class af scheduler Q2;
        forwarding-class nc scheduler Q3;
        forwarding-class ef1 scheduler Q4;
        forwarding-class ef2 scheduler Q5;
        forwarding-class af1 scheduler Q6;
        forwarding-class nc1 scheduler Q7;
    }
}
schedulers {
    Q0 {
        transmit-rate percent 25;
        buffer-size percent 25;
        priority low;
        drop-profile-map loss-priority any protocol both drop-default;
    }
    Q1 {
        buffer-size temporal 2000;
        priority strict-high;
        drop-profile-map loss-priority any protocol both drop-ef;
    }
    Q2 {
        transmit-rate percent 35;
        buffer-size percent 35;
        priority low;
        drop-profile-map loss-priority any protocol both drop-default;
    }
    Q3 {
        transmit-rate percent 5;
        buffer-size percent 5;
        drop-profile-map loss-priority any protocol both drop-default;
    }
    Q4 {
        transmit-rate percent 5;
        priority high;
        drop-profile-map loss-priority any protocol both drop-ef;
    }
}

```

```

    Q5 {
        transmit-rate percent 10;
        priority high;
        drop-profile-map loss-priority any protocol both drop-ef;
    }
    Q6 {
        transmit-rate remainder;
        priority low;
        drop-profile-map loss-priority any protocol both drop-default;
    }
    Q7 {
        transmit-rate percent 5;
        priority high;
        drop-profile-map loss-priority any protocol both drop-default;
    }
}

```

**Configuring a Scheduler Map Applicable to an Interface Restricted to Four Queues**

For PICs restricted to four queues, if you map multiple forwarding classes to a queue, the multiple forwarding classes must refer to the same scheduler.

```

[edit class-of-service]
scheduler-maps {
    interface-restricted {
        forwarding-class be scheduler Q0;
        forwarding-class ef scheduler Q1;
        forwarding-class ef1 scheduler Q1;
        forwarding-class ef2 scheduler Q1;
        forwarding-class af1 scheduler Q2;
        forwarding-class af scheduler Q2;
        forwarding-class nc scheduler Q3;
        forwarding-class nc1 scheduler Q3;
    }
}
[edit class-of-service]
restricted-queues {
    forwarding-class be queue 0;
    forwarding-class ef queue 1;
    forwarding-class ef1 queue 1;
    forwarding-class ef2 queue 1;
    forwarding-class af queue 2;
    forwarding-class af1 queue 2;
    forwarding-class nc queue 3;
    forwarding-class nc1 queue 3;
}

```

### Configuring an IP Precedence Classifier and Rewrite Tables

```
[edit class-of-service]
classifiers {
  inet-precedence inet-classifier {
    forwarding-class be {
      loss-priority low code-points 000;
    }
    forwarding-class af11 {
      loss-priority high code-points 001;
    }
    forwarding-class ef {
      loss-priority low code-points 010;
    }
    forwarding-class nc1 {
      loss-priority high code-points 011;
    }
    forwarding-class {
      loss-priority low code-points 100;
    }
    forwarding-class af12 {
      loss-priority high code-points 101;
    }
    forwarding-class ef1 {
      loss-priority low code-points 110;
    }
    forwarding-class nc2 {
      loss-priority high code-points 111;
    }
  }
}
exp exp-rw-table {
  forwarding-class be {
    loss-priority low code-point 000;
  }
  forwarding-class af11 {
    loss-priority high code-point 001;
  }
  forwarding-class ef {
    loss-priority low code-point 010;
  }
  forwarding-class nc1 {
    loss-priority high code-point 111;
  }
  forwarding-class be1 {
    loss-priority low code-point 100;
  }
  forwarding-class af12 {
    loss-priority high code-point 101;
  }
  forwarding-class ef1 {
    loss-priority low code-point 110;
  }
  forwarding-class nc2 {
    loss-priority low code-point 111;
  }
}
```

```

inet-precedence inet-rw-table {
  forwarding-class be {
    loss-priority low code-point 000;
  }
  forwarding-class af11 {
    loss-priority high code-point 001;
  }
  forwarding-class ef1 {
    loss-priority low code-point 010;
  }
  forwarding-class nc1 {
    loss-priority low code-point 111;
  }
  forwarding-class be1 {
    loss-priority low code-point 100;
  }
  forwarding-class af12 {
    loss-priority high code-point 101;
  }
  forwarding-class ef1 {
    loss-priority low code-point 111;
  }
  forwarding-class nc2 {
    loss-priority low code-point 110;
  }
}

```

## Classifying Packets by Behavior Aggregate

---

By default, packet classification is performed by IP precedence and MPLS EXP classifiers. The IP precedence classifier handles all incoming IP packets, and the MPLS EXP classifier handles all incoming MPLS packets.

You can configure the following classifier types:

DSCP, DSCP IPv6, or IP precedence—IP packet classification (Layer 3 headers)

MPLS EXP—MPLS packet classification (Layer 2 headers)

IEEE 802.1p—Packet classification (Layer 2 headers)

If you apply an IEEE 802.1 classifier to a logical interface, this classifier takes precedence and is not compatible with any other classifier type. Classifiers for IP (DSCP or IP precedence) and MPLS (EXP) can coexist on a logical interface if the hardware platform requirements are met. (See Table 51 on page 833).

The simplest way to classify a packet is to use behavior aggregate classification. The DSCP, DSCP IPv6, or IP precedence bits of the IP header convey the behavior aggregate class information. The information might also be found in the MPLS EXP bits or IEEE 802.1p CoS bits.

Table 50 shows the default system classification scheme for the well-known DSCPs.

**Table 50: Default Behavior Aggregate Classification**

DSCP and DSCP IPv6	Forwarding Class	PLP
ef	expedited-forwarding	low
af11	assured-forwarding	low
af12	assured forwarding	high
af13	assured forwarding	high
af21	best-effort	low
af22	best-effort	low
af23	best-effort	low
af31	best-effort	low
af32	best-effort	low
af33	best-effort	low
af41	best-effort	low
af42	best-effort	low
af43	best-effort	low
be	best-effort	low
cs1	best-effort	low
cs2	best-effort	low
cs3	best-effort	low
cs4	best-effort	low
cs5	best-effort	low
nc1/cs6	network-control	low
nc2/cs7	network control	low
other	best-effort	low

All af classes other than af1X are mapped to best-effort, since RFC 2597 prohibits a node from aggregating classes. In effect, mapping to best-effort implies that the node does not support that class.

To define new classifiers for all code-point types, include the classifiers statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
classifiers {
  (dscp | dscp-ipv6 | exp | ieee-802.1 | inet-precedence) classifier-name {
    import [classifier-name | default];
    forwarding-class class-name {
      loss-priority level {
        code-points [ aliases ] [ 6-bit-patterns ];
      }
    }
  }
}
```

The map sets the forwarding class and PLP for a specific set of code-point aliases and bit patterns. The inputs of the map are code-point aliases and bit patterns. The outputs of the map are the forwarding class and the PLP. For more information about how CoS maps work, see Table 45 on page 804.

The classifiers work as follows:

dscp—Handles incoming IPv4 packets.

dscp-ipv6—Handles incoming IPv6 packets. For more information, see “Applying DSCP IPv6 Classifiers” on page 834.

exp—Handles MPLS packets using Layer 2 headers.

ieee-802.1—Handles Layer 2 CoS.

inet-precedence—Handles incoming IPv4 packets. IP precedence mapping requires only the upper three bits of the DSCP field.

A classifier takes a specified bit pattern as either the literal pattern or as a defined alias and attempts to match it to the type of packet arriving on the interface. If the information in the packet’s header matches the specified pattern, the packet is sent to the appropriate queue, defined by the forwarding class associated with the classifier.

The code-point aliases and bit patterns are the input for the map. The loss priority and forwarding class are outputs of the map. In other words, the map sets the PLP and forwarding class for a given set of code-point aliases and bit patterns.

You can use any table, including the default, in the definition of a new classifier by including the `import` statement. The imported classifier is used as a template and is not modified. Whenever you commit a configuration that assigns a new *class-name* and *loss-priority* value to a code-point alias or set of bits, it replaces that entry in the imported classifier template. As a result, you must explicitly specify every code point in every designation that requires modification.



**NOTE:** If an interface is mounted on an M-series FPC, you can apply to the interface the default exp classifier only. If an interface is mounted on an enhanced FPC, you can create a new exp classifier and apply it to an interface.

On M320 and T-series platforms, the behavior aggregate (BA) classifier loss priority setting has no effect. For more information, see “Overriding the Default PLP on M320 and T-series Platforms” on page 834.

---

This section is organized as follows:

Applying a Classifier to a Logical Interface on page 833

Applying DSCP IPv6 Classifiers on page 834

Overriding the Default PLP on M320 and T-series Platforms on page 834

## Applying a Classifier to a Logical Interface

You can apply the classification map to a logical interface by including the `classifiers` statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy level:

```
[edit class-of-service interfaces interface-name unit logical-unit-number]
classifiers (dscp | dscp-ipv6 | exp | ieee-802.1 | inet-precedence) (classifier-name
| default);
```

You can use interface wildcards for *interface-name* and *logical-unit-number*.



**NOTE:** If you apply an IEEE 802.1p classifier to a logical interface, you cannot apply non-IEEE classifiers on other logical interfaces on the same physical interface.

Table 51 shows the classifiers you can combine on a single logical interface, by platform.

**Table 51: Supported Classifier Combinations by Platform**

Classifier Combinations	T-series and M320	Other M-series with Regular FPCs	Other M-series with Enhanced FPCs
dscp and inet-precedence	No	No	No
dscp-ipv6 and (dscp   inet-precedence)	Yes	No	No
exp and ieee 802.1	No	No	No
ieee 802.1 and (dscp   dscp-ipv6   exp   inet-precedence)	No	No	Yes
exp and (dscp   dscp-ipv6   inet-precedence)	Yes	No	Yes

## Applying DSCP IPv6 Classifiers

For T-series platforms, you can apply separate classifiers for IPv4 and IPv6 packets per logical interface by including the classifiers statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy level and specifying the dscp and dscp-ipv6 classifier types:

```
[edit class-of-service interfaces interface-name unit logical-unit-number]
  classifiers dscp (classifier-name | default);
  classifiers dscp-ipv6 (classifier-name | default);
```

For M-series enhanced FPCs, you cannot apply separate classifiers for IPv4 and IPv6 packets on a single logical interface. Instead, classifier assignment works as follows:

If you assign a DSCP classifier only, IPv4 and IPv6 packets are classified using the DSCP classifier.

If you assign an IP precedence classifier only, IPv4 and IPv6 packets are classified using the IP precedence classifier. In this case, the lower three bits of the DSCP field are ignored because IP precedence mapping requires the upper three bits only.

If you assign either the DSCP or the IP precedence classifier in conjunction with the DSCP IPv6 classifier, the commit fails.

If you assign a DSCP IPv6 classifier only, IPv4 and IPv6 packets are classified using the DSCP IPv6 classifier, but the commit displays a warning message.

For more information, see Table 51 on page 833.

## Overriding the Default PLP on M320 and T-series Platforms

By default, the least significant bit of the code point sets the packet loss priority (PLP) value. For example, code point 000 is associated with PLP low, and code point 001 is associated with PLP high. In general, you can override this default by configuring a BA classifier, as discussed in “Classifying Packets by Behavior Aggregate” on page 830.



**NOTE:** On M320 and T-series platforms, you configure the rewrite rule for loss-priority high by setting the least-significant bit or by setting loss-priority high within a multifield classifier. If the least-significant bit is set, the loss-priority is high, regardless how you configure the multifield classifier. However, if you do not configure the least-significant bit, you can overwrite the loss priority using a multifield classifier.

For more information about MF classifiers, see the *JUNOS Policy Framework Configuration Guide*.

---

**Example: Overriding the Default PLP on M320 and T-series Platforms**

Override the default PLP:

1. The least-significant bit of the code point is 1; therefore, the loss priority is high on M320 and T-series platforms. The loss priority is set to low for the code point 111; however, on M320 and T-series platforms, this loss-priority setting has no effect.

```
[edit class-of-service]
classifiers {
  dscp ba-classifier {
    forwarding-class expedited-forwarding {
      loss-priority low code-points 111;
    }
  }
}
```

2. For M320 and T-series platforms, this MF classifier overrides the default.

```
[edit firewall filter ef-filter term ef-multifield]
then {
  forwarding-class expedited-forwarding;
  loss-priority low;
}
```

## Configuring Scheduling Maps

---

You use *schedulers* to configure transmission scheduling and rate control parameters. Schedulers define the priority, bandwidth, delay buffer size, rate control status, and RED drop profiles to be applied to a particular class of traffic.

You associate the schedulers with forwarding classes by means of scheduler maps. You can then associate each scheduler map with an interface, thereby configuring the hardware queues, packet schedulers, and RED processes that operate according to this mapping.

A scheduler configuration block specifies the buffer size, bandwidth, and priority for a queue. It also specifies the RED drop profile for packets that fall within specification and out of specification.

Physical interfaces (for example, t3-0/0/0, t3-0/0/0:0, and ge-0/0/0) support scheduling with any encapsulation type pertinent to that physical interface. For a single port, you cannot apply scheduling to the physical interface if you have applied scheduling to one or more of the associated logical interfaces.

Logical interfaces (for example, t3-0/0/0.0 and ge-0/0/0.0) support scheduling on DLCIs or VLANs only. Furthermore, logical interface scheduling is not supported on PICs that do not have IQ. .



**NOTE:** In the JUNOS software implementation, the term *logical interfaces* generally refers to interfaces you configure by including the unit statement at the [edit interfaces *interface-name*] hierarchy level. As such, logical interfaces have the *.logical* descriptor at the end of the interface name, as in ge-0/0/0.1 or t1-0/0/0:0.1, where the logical unit number is 1.

Although channelized interfaces are generally thought of as logical or virtual, the JUNOS software sees T3, T1, and NxDS0 interfaces within a channelized IQ PIC as physical interfaces. For example, both t3-0/0/0 and t3-0/0/0:1 are treated as physical interfaces by the JUNOS software. In contrast, t3-0/0/0.2 and t3-0/0/0:1.2 are considered logical interfaces because they have the .2 at the end of the interface names.

Within the [edit class-of-service] hierarchy level, you cannot use the *.logical* descriptor when you assign properties to logical interfaces. Instead, you must include the unit statement in the configuration. For example:

```
[edit class-of-service]
user@host# set interfaces t3-0/0/0 unit 0 scheduler-map map1
```

To configure schedulers, include the schedulers statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
schedulers {
    buffer-size (percent percentage | remainder | temporal microseconds);
    drop-profile-map loss-priority (any | high | low) protocol (any | non-tcp | tcp)
        drop-profile profile-name;
    priority priority-level;
    transmit-rate (rate | percent percentage | remainder) <exact>;
}
```

This section is organized as follows:

Default Scheduler on page 837

Configuring the Scheduler Buffer Size on page 837

Configuring the Scheduler Drop Profile on page 838

Configuring Priority Scheduling on page 838

Configuring the Scheduler Transmission Rate on page 840

Configuring the Scheduler Map on page 841

Associating the Scheduler Map with an Output Interface on page 841

Associating a Scheduler Map with a DLCI or VLAN on page 846

Associating a Scheduler with a Fabric Priority on page 851

## Default Scheduler

The following default scheduler is provided with the installation. These settings are not visible in the output of the show class-of-service command; rather, they are implicit.

```
[edit class-of-service]
schedulers {
  network-control {
    transmit-rate percent 5;
    buffer-size percent 5;
    priority low;
    drop-profile-map loss-priority any protocol any drop-profile terminal;
  }
  best-effort {
    transmit-rate percent 95;
    buffer-size percent 95;
    priority low;
    drop-profile-map loss-priority any protocol any drop-profile terminal;
  }
}
drop-profiles {
  terminal {
    fill-level 100 drop-probability 100;
  }
}
```

## Configuring the Scheduler Buffer Size

To control congestion at the output stage, you can configure the delay-buffer bandwidth. The delay-buffer bandwidth provides packet buffer space to absorb burst traffic up to the specified duration of delay. Once the specified delay buffer becomes full, packets with 100 percent drop probability are dropped from the head of the buffer.

By default, the buffer sizes for queues 0 through 7 are 95, 0, 0, 5, 0, 0, 0, and 0 percent of the total available buffer space.

To configure the buffer size, include the buffer-size statement at the [edit class-of-service schedulers *scheduler-name*] hierarchy level:

```
[edit class-of-service schedulers scheduler-name]
buffer-size (percent percentage | remainder | temporal microseconds);
```

For each scheduler, you can configure the buffer size as one of the following:

A percentage of the total buffer

The remaining buffer available. The remainder is the buffer percentage that is not assigned to other queues. For example, if you assign 40 percent of the delay buffer to queue 0, allow queue 3 to keep the default allotment of 5 percent, and assign the remainder to queue 7, then queue 7 uses approximately 55 percent of the delay buffer.

A temporal value, in microseconds. For the temporal setting, the queuing algorithm starts dropping packets when it queues more than a computed number of bytes. This maximum is computed by multiplying the logical interface speed by the configured temporal value.

The buffer size temporal value differs by platform type. For T-series and M320 routing platforms, the temporal value can be from 1 through 50,000 microseconds. For other M-series routing platforms, the temporal value can be from 1 through 200,000 microseconds. For IQ PICs on other M-series routing platforms, the temporal value can be from 1 through 100,000 microseconds. For IQ PICs on T-series and M320 platforms, the 1 through 50,000 microseconds limitation applies.

### **Configuring the Scheduler Drop Profile**

The scheduler drop profile defines the drop probabilities across the range of delay-buffer occupancy, thereby supporting the RED process. Depending on the drop probabilities, RED might drop packets aggressively long before the buffer becomes full, or it might drop only a few packets even if the buffer is almost full.

By default, the drop profile is mapped to packets with low PLP, regardless of protocol type. To configure which packet types are mapped to a specified drop profile, include the drop-profile-map statement at the [edit class-of-service schedulers *scheduler-name*] hierarchy level:

```
[edit class-of-service schedulers scheduler-name]
  drop-profile-map loss-priority (any | high | low) protocol (any | non-tcp | tcp)
    drop-profile profile-name;
```

The map sets the drop profile for a specific PLP and protocol type. The inputs for the map are the PLP and the protocol type. The output is the drop profile. For more information about how CoS maps work, see Table 45 on page 804.

For each scheduler, you can configure four separate drop profiles, one for each combination of loss priority (low or high) and IP transport protocol (TCP/IP or non-TCP/IP).

You can configure a maximum of 32 different drop profiles.

For more information, see “Configuring RED Drop Profiles” on page 853.

### **Configuring Priority Scheduling**

Priority scheduling determines the order in which an output interface transmits traffic from the queues. The JUNOS software supports multiple levels of transmission priority, which in order of increasing priority are low, medium-low, medium-high, and high.

The JUNOS software priorities map to numeric priorities in the underlying hardware. Two software priorities behave differently only if they map to two distinct hardware priorities. For more information, see “Hardware Priority Mappings” on page 840.

To configure priority scheduling, include the priority statement at the [edit class-of-service schedulers *scheduler-name*] hierarchy level:

```
[edit class-of-service schedulers scheduler-name]
priority priority-level;
```

Higher-priority queues transmit packets ahead of lower priority queues as long as the higher-priority forwarding classes retain enough bandwidth credit. When you configure a higher-priority queue with a significant fraction of the transmission bandwidth, the queue might lock out lower priority traffic.

You can also configure one queue per interface to have strict-high priority, which works the same as high priority, but provides unlimited transmission bandwidth. As long as the queue with strict-high priority has traffic to send, it receives precedence over all other queues, except queues with high priority. Queues with strict-high and high priority take turns transmitting packets until the strict-high queue is empty, the high priority queues are empty, or the high priority queues run out of bandwidth credit. Only then can lower priority queues send traffic.

The high priority allows you to protect traffic classes from being starved by traffic in a strict-high queue. For example, a network-control queue might require a small bandwidth allocation (say, 5 percent). You can assign high priority to this queue to prevent it from being underserved.



**NOTE:** When you configure a queue to have strict-high priority, you should not include the transmit-rate statement in the queue configuration because the transmission rate of a strict-high priority queue is not limited by the WRR configuration.

### Example: Configuring Priority Scheduling

Configure priority scheduling, as shown in the following example:

1. Configure a scheduler, be-sched, with medium-low priority.

```
[edit class-of-service]
schedulers {
  be-sched {
    priority medium-low;
  }
}
```

2. Configure a scheduler map, be-map, that associates be-sched with the best-effort forwarding class.

```
[edit class-of-service]
scheduler-maps {
  be-map {
    forwarding-class best-effort scheduler be-sched;
  }
}
```

3. Assign be-map to a Gigabit Ethernet interface, ge-0/0/0.

```
[edit class-of-service]
interfaces {
  ge-0/0/0 {
    scheduler-map be-map;
  }
}
```

```

    }
}

```

### Hardware Priority Mappings

The priority levels you configure map to hardware priority levels. These priority mappings differ depending on the FPC type in which the PIC is mounted.

Table 52 shows the priority mappings by FPC type. Note, for example, that on M320 FPCs and T-series enhanced FPCs, the software priorities medium-low and medium-high behave similarly because they map to the same hardware priority level.

**Table 52: Scheduling Priority Mappings by FPC Type**

Priority Levels	Mappings for FPCs	Mappings for M320 FPCs and T-series Enhanced FPCs
low	0	0
medium-low	0	1
medium-high	1	1
high	1	2
strict-high (full interface bandwidth)	1	2

### Configuring the Scheduler Transmission Rate

The transmission rate control determines the actual traffic bandwidth from each of the forwarding classes you configure. The rate is specified in bits per second. You can limit the transmission bandwidth to the exact value you configure, or allow it to exceed the configured rate if additional bandwidth is available from other queues.

For 8-port, 12-port, and 48-port Fast Ethernet PICs, transmission scheduling is not supported.

By default, the transmission rates for queues 0 through 7 are 95, 0, 0, 5, 0, 0, 0, and 0 percent. To configure transmission scheduling, include the `transmit-rate` statement at the `[edit class-of-service schedulers scheduler-name]` hierarchy level:

```

[edit class-of-service schedulers scheduler-name]
  transmit-rate (rate | percent percentage | remainder) <exact>;

```

You can specify the transmit rate as follows:

*rate*—Transmission rate, in bits per second. The rate can be from 3200 through 32,000,000,000 bits per second (bps).

percent *percentage*—Percentage of transmission capacity.

remainder—Use remaining rate available. In the configuration, you cannot combine the remainder and exact options.

exact—Enforce the exact transmission rate or percentage you configure with the transmit-rate *rate* or transmit-rate percent statement. Under sustained congestion, a rate-controlled queue that goes into negative credit fills up and eventually drops packets. You specify the exact option as follows:

```
[edit class-of-service schedulers scheduler-name]
transmit-rate rate exact;
```

```
[edit class-of-service schedulers scheduler-name]
transmit-rate percent percent exact;
```

In the configuration, you cannot combine the remainder and exact options.

### Configuring the Scheduler Map

Once you define a scheduler, you can include it in a *scheduler map*, which maps a specified forwarding class to a scheduler configuration. To do this, include the scheduler-maps statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
scheduler-maps {
  map-name {
    forwarding-class class-name scheduler scheduler-name;
  }
}
```

### Associating the Scheduler Map with an Output Interface

After you have defined the scheduler map, as described in “Configuring the Scheduler Map” on page 841, you can associate it with an output interface. To do this, include the scheduler-map statement at the [edit class-of-service interfaces *interface-name*] hierarchy level:

```
[edit class-of-service interfaces interface-name]
scheduler-map map-name;
```

Interface wildcards are supported.

Generally, you can associate schedulers with physical interfaces only. For some IQ interfaces, you can also associate schedulers with the logical interface. For more information, see “Associating a Scheduler Map with a DLCI or VLAN” on page 846.



**NOTE:** For original Channelized OC12 PICs, limited CoS functionality is supported. For more information, contact Juniper Networks customer support.

## Scheduling Packet Forwarding Component Queues

On IQ interfaces, the traffic that is fed from the packet forwarding components into the PIC uses low PLP by default and is distributed evenly across the four chassis queues (not PIC queues), regardless of the scheduling configuration for each logical interface.

You can avoid congestion by controlling the aggregated traffic transmitted from the chassis queues into the PIC. To do so, you can configure the chassis queues to derive their scheduling configuration from the associated logical interfaces, or you can assign a custom scheduler to the chassis queues.

To configure the chassis queues to derive their scheduling configuration from the associated logical interface scheduling configuration, include the `scheduler-map-chassis` derived statement at the `[edit class-of-service interfaces interface-name]` hierarchy level:

```
[edit class-of-service interfaces type-fpc/pic/*]
scheduler-map-chassis derived;
```

For information about logical interface scheduling configuration, see “Associating a Scheduler Map with a DLCI or VLAN” on page 846.

To assign a custom scheduler to the packet forwarding component queues, include the `scheduler-map-chassis` statement at the `[edit class-of-service interfaces interface-name]` hierarchy level:

```
[edit class-of-service interfaces type-fpc/pic/*]
scheduler-map-chassis map-name;
```

For information about defining the scheduler map referenced by `map-name`, see “Configuring the Scheduler Map” on page 841.

Generally, when you include the `scheduler-map-chassis` statement in the configuration, you must use an interface wildcard for the interface name, as in `type-fpc/pic/*`. The wildcard must use this per-PIC format—for example, `so-1/2/*`, which means all interfaces on FPC slot 1, PIC slot 2.

One exception to this rule is that for the Gigabit Ethernet IQ PIC only, you can apply the chassis scheduler map to a single interface. For PICs that support both the per-interface and per-PIC format (currently, only the Gigabit Ethernet IQ PIC) we recommend that you use either the per-interface or the per-PIC format, but not both.

According to JUNOS software wildcard rules, specific interface configurations override wildcard configurations. For chassis scheduler map configuration, this rule does not apply; instead, specific interface CoS configurations are added to the chassis scheduler map configuration. For more information about how wildcards work with chassis scheduler maps, see “Example: Associating a Scheduler Map with Packet Forwarding Component Queues” on page 843. For general information about wildcards, see the *JUNOS System Basics Configuration Guide*.

You can include both the scheduler-map and the scheduler-map-chassis statements in the same interface configuration. The scheduler-map statement controls the scheduler inside the PIC, while the scheduler-map-chassis statement controls the aggregated traffic transmitted into the entire PIC. For more information about the scheduler-map statement, see “Associating the Scheduler Map with an Output Interface” on page 841.

### Example: Associating a Scheduler Map with Packet Forwarding Component Queues

#### Associating a Chassis Scheduler Map with a 2-Port IQ PIC

The two interfaces are so-0/1/0 and so-0/1/1.

According to customary wildcard rules, the so-0/1/0 configuration overrides the so-0/1/\* configuration, implying that the chassis scheduler map MAP1 is not applied to so-0/1/0. However, the wildcard rule is not obeyed in this case, meaning that the chassis scheduler map applies to both interfaces so-0/1/0 and so-0/1/1.

```
[edit interfaces]
so-0/1/0 {
  unit 0 {
    classifiers {
      inet-precedence default;
    }
  }
}
so-0/1/* {
  scheduler-map-chassis MAP1;
}
```

#### Not Recommended: Gigabit Ethernet IQ Example

On a Gigabit Ethernet IQ PIC, you can apply the chassis scheduler map at both the per-interface level and the wildcard level. We do not recommend this because the wildcard chassis scheduler map takes precedence, which might not be the desired effect. For example, if you want to apply MAP1 to port 0 and MAP2 to port 1, the following is not recommended.

```
[edit class-of-service]
interfaces {
  ge-0/1/0 {
    scheduler-map-chassis MAP1;
  }
  ge-0/1/* {
    scheduler-map-chassis MAP2;
  }
}
```

#### Recommended: Gigabit Ethernet IQ Example

Instead, we recommend this:

```
[edit class-of-service]
interfaces {
  ge-0/1/0 {
    scheduler-map-chassis MAP1;
  }
  ge-0/1/1 {
    scheduler-map-chassis MAP2;
  }
}
```

**Configuring ATM CoS with a Normal Scheduler and a Chassis Scheduler**

For more information about ATM CoS, see “Configuring ATM2 IQ VC Tunnel CoS Components” on page 236.

```
[edit class-of-service]
interfaces {
  at-1/2/* {
    scheduler-map-chassis map-1;
  }
}

[edit interfaces]
at-1/2/0 {
  atm-options {
    vpi 0;
    linear-red-profiles red-profile-1 {
      queue-depth 35000 high-plp-threshold 75 low-plp-threshold 25;
    }
    scheduler-maps map-1 {
      vc-cos-mode strict;
      forwarding-class best-effort {
        priority low;
        transmit-weight percent 25;
        linear-red-profile red-profile-1;
      }
    }
  }
  unit 0 {
    vci 0.128;
    shaping {
      vbr peak 20m sustained 10m burst 20;
    }
    atm-scheduler-map map-1;
    family inet {
      address 192.168.0.100/32 {
        destination 192.168.0.101;
      }
    }
  }
}
```

**Configuring Two T3 Interfaces on a Channelized DS3 IQ PIC**

```
[edit interfaces]
ct3-3/0/0 {
  no-partition interface-type t3; # use entire port 0 as T3
}
ct3-3/0/1 {
  no-partition interface-type t3; # use entire port 1 as T3
}
t3-3/0/0 {
  unit 0 {
    family inet {
      address 10.0.100.1/30;
    }
  }
}
```

```
t3-3/0/1 {
  unit 0 {
    family inet {
      address 10.0.101.1/30;
    }
  }
}
```

### Associating Normal Schedulers with the Two T3 Interfaces

Configure a scheduler for the aggregated traffic transmitted into both T3 interfaces.

```
[edit class-of-service]
interfaces {
  t3-3/0/0 {
    scheduler-map sched-qct3-0;
  }
  t3-3/0/1 {
    scheduler-map sched-qct3-1;
  }
}
scheduler-maps {
  sched-qct3-0 {
    forwarding-class best-effort scheduler be-qct3-0;
    forwarding-class expedited-forwarding scheduler ef-qct3-0;
    forwarding-class assured-forwarding scheduler as-qct3-0;
    forwarding-class network-control scheduler nc-qct3-0;
  }
  sched-qct3-1 {
    forwarding-class best-effort scheduler be-qct3-1;
    forwarding-class expedited-forwarding scheduler ef-qct3-1;
    forwarding-class assured-forwarding scheduler as-qct3-1;
    forwarding-class network-control scheduler nc-qct3-1;
  }
  sched-chassis-to-q {
    forwarding-class best-effort scheduler be-chassis;
    forwarding-class expedited-forwarding scheduler ef-chassis;
    forwarding-class assured-forwarding scheduler as-chassis;
    forwarding-class network-control scheduler nc-chassis;
  }
}
schedulers {
  be-qct3-0 {
    transmit-rate percent 40;
  }
  ef-qct3-0 {
    transmit-rate percent 30;
  }
  as-qct3-0 {
    transmit-rate percent 20;
  }
  nc-qct3-0 {
    transmit-rate percent 10;
  }
  ...
}
```

**Associating a Chassis Scheduler with the Two T3 Interfaces** Bind a scheduler to the aggregated traffic transmitted into the entire PIC. The chassis scheduler controls the traffic from the packet forwarding components feeding the interface t3-3/0/\*.

```
[edit class-of-service]
interfaces {
  t3-3/0/* {
    scheduler-map-chassis sched-chassis-to-q;
  }
}
```

### **Associating a Scheduler Map with a DLCI or VLAN**

*Logical interface scheduling* allows you to associate schedulers with interfaces you configure at the [edit interfaces *interface-name* unit *logical-unit-number*] hierarchy level.

By default, transmission scheduling is not enabled on logical interfaces. You can enable logical interface transmission scheduling on the following PICs:

Channelized OC12 IQ PIC

Channelized STM1 IQ PIC

Channelized T3 IQ PIC

Channelized E1 IQ PIC

E3 IQ PIC

Gigabit Ethernet IQ PIC

Physical interfaces (for example, t3-0/0/0, t3-0/0/0:0, and ge-0/0/0) support scheduling with any encapsulation type pertinent to that physical interface. For a single port, you cannot apply scheduling to the physical interface if you apply scheduling to one or more of the associated logical interfaces.

Logical interfaces (for example, t3-0/0/0.0, ge-0/0/0.0, and t1-0/0/0:0.1) support scheduling on DLCIs or VLANs only. Furthermore, logical interface scheduling is not supported on PICs that do not have IQ. .



**NOTE:** In the JUNOS software implementation, the term *logical interfaces* generally refers to interfaces you configure by including the unit statement at the [edit interfaces *interface-name*] hierarchy level. As such, logical interfaces have the *.logical* descriptor at the end of the interface name, as in ge-0/0/0.1 or t1-0/0/0:0.1, where the logical unit number is 1.

Although channelized interfaces are generally thought of as logical or virtual, the JUNOS software sees T3, T1, and NxDS0 interfaces within a channelized IQ PIC as physical interfaces. For example, both t3-0/0/0 and t3-0/0/0:1 are treated as physical interfaces by the JUNOS software. In contrast, t3-0/0/0.2 and t3-0/0/0:1.2 are considered logical interfaces because they have the .2 at the end of the interface names.

Within the [edit class-of-service] hierarchy level, you cannot use the *.logical* descriptor when you assign properties to logical interfaces. Instead, you must include the unit statement in the configuration. For example:

```
[edit class-of-service]
user@host#set interfaces t3-0/0/0 unit 0 scheduler-map map1
```

Table 53 shows the interfaces that support transmission scheduling.

**Table 53: Transmission Scheduling Support by Interfaces Type**

Interface Type	Supported	Examples
<b>IQ PICs</b>		
Physical interfaces	Yes	Example of supported configuration, configured on ATM2 IQ PIC: [edit class-of-service interfaces at-0/0/0] scheduler-map map-1;
Channelized interfaces configured on IQ PICs	Yes	Example of supported configuration, configured on Channelized DS3 IQ PIC: [edit class-of-service interfaces t1-0/0/0:1] scheduler-map map-1;
Logical interfaces (DLCIs and VLANs only) configured on IQ PICs	Yes	Examples of supported configurations: Configured on Gigabit Ethernet IQ PIC with VLAN tagging enabled: [edit class-of-service interfaces ge-0/0/0 unit 1] scheduler-map map-1; Configured on E3 IQ PIC with Frame Relay encapsulation: [edit class-of-service interfaces e3-0/0/0 unit 1] scheduler-map map-1; Configured on Channelized T3 IQ PIC with Frame Relay encapsulation: [edit class-of-service interfaces t1-0/0/0 unit 1] scheduler-map map-1;

Interface Type	Supported	Examples
Logical interfaces configured on IQ PICs (Interfaces that are not DLCIs or VLANs)	No	<p>Example of unsupported configurations:</p> <p>Configured on E3 IQ PIC with Cisco HDLC encapsulation:                      [edit class-of-service interfaces e3-0/0/0 unit 1]                      scheduler-map map-1;</p> <p>Configured on ATM2 IQ PIC with LLC/SNAP encapsulation:                      [edit class-of-service interfaces at-0/0/0 unit 1]                      scheduler-map map-1;</p> <p>Configured on Channelized OC12 IQ PIC with PPP encapsulation:                      [edit class-of-service interfaces t1-0/0/0:1 unit 1]                      scheduler-map map-1;</p>
<b>Non-IQ PICs</b>		
Physical interfaces	Yes	<p>Example of supported configuration, configured on T3 PIC:                      [edit class-of-service interfaces t3-0/0/0]                      scheduler-map map-1;</p>
Channelized OC12 PIC	Yes	<p>Example supported configuration, configured on Channelized OC12 PIC:                      [edit class-of-service interfaces t3-0/0/0:1]                      scheduler-map map-1;</p>
Channelized interfaces (except the Channelized OC12 PIC)	No	<p>Example unsupported configuration, configured on Channelized STM1 PIC:                      [edit class-of-service interfaces e1-0/0/0:1]                      scheduler-map map-1;</p>
Logical interfaces	No	<p>Examples of unsupported configurations:</p> <p>Configured on Fast Ethernet PIC:                      [edit class-of-service interfaces fe-0/0/0 unit 1]                      scheduler-map map-1;</p> <p>Configured on Gigabit Ethernet PIC:                      [edit class-of-service interfaces ge-0/0/0 unit 0]                      scheduler-map map-1;</p> <p>Configured on ATM1 PIC:                      [edit class-of-service interfaces at-0/0/0 unit 2]                      scheduler-map map-1;</p> <p>Configured on Channelized OC12 PIC:                      [edit class-of-service interfaces t3-0/0/0:0 unit 2]                      scheduler-map map-1;</p>

To configure transmission scheduling on logical interfaces, perform the following steps:

1. Enable scheduling on the interface by including the per-unit-scheduler statement at the [edit interfaces *interface-name*] hierarchy level:

```
[edit interfaces interface-name]  
per-unit-scheduler;
```

2. Associate a scheduler with the interface by including the scheduler-map statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy level:

```
[edit class-of-service interfaces interface-name unit logical-unit-number]  
scheduler-map map-name;
```

3. Configure shaping on the interface by including the shaping-rate statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy level:

```
[edit class-of-service interfaces interface-name unit logical-unit-number]  
shaping-rate rate;
```

By default, the logical interface bandwidth is the average of unused bandwidth for the number of logical interfaces that require default bandwidth treatment. You can specify a peak bandwidth rate in bps, either as a complete decimal number or as a decimal number followed by the abbreviation k (1000), m (1,000,000), or g (1,000,000,000). The range is from 1000 through 32,000,000,000 bps.

The sum of the bandwidth you allocate to all logical interfaces on a physical interface must not exceed the bandwidth of the physical interface.

**Example: Associating a Scheduler Map name with a DLCI or VLAN**

Associate the scheduler `sched-map-logical-0` with logical interface unit 0 on physical interface `t3-1/0/0`, and allocate 10 megabits per second (Mbps) of transmission bandwidth to the logical interface.

Associate the scheduler `sched-map-logical-1` with logical interface unit 1 on physical interface `t3-1/0/0`, and allocate 20 Mbps of transmission bandwidth to the logical interface.

The allocated bandwidth is shared among the individual forwarding classes in the scheduler map. Although these schedulers are configured on a single physical interface, they are independent from each other. Traffic on one logical interface unit does not affect the transmission priority, bandwidth allocation, or drop behavior on the other logical interface unit.

```
[edit interfaces]
t3-1/0/0:1 {
  encapsulation frame-relay;
  per-unit-scheduler;
}

[edit class-of-service]
interfaces {
  t3-1/0/0:1 {
    unit 0 {
      scheduler-map sched-map-logical-0;
      shaping-rate 10m;
    }
    unit 1 {
      scheduler-map sched-map-logical-1;
      shaping-rate 20m;
    }
  }
}

scheduler-maps {
  sched-map-logical-0 {
    forwarding-class best-effort scheduler sched-best-effort-0;
    forwarding-class assured-forwarding scheduler sched-bronze-0;
    forwarding-class expedited-forwarding scheduler sched-silver-0;
    forwarding-class network-control scheduler sched-gold-0;
  }
  sched-map-logical-1 {
    forwarding-class best-effort scheduler sched-best-effort-1;
    forwarding-class assured-forwarding scheduler sched-bronze-1;
    forwarding-class expedited-forwarding scheduler sched-silver-1;
    forwarding-class network-control scheduler sched-gold-1;
  }
}
```

```

schedulers {
  sched-best-effort-0 {
    transmit-rate 4m;
  }
  sched-bronze-0 {
    transmit-rate 3m;
  }
  sched-silver-0 {
    transmit-rate 2m;
  }
  sched-gold-0 {
    transmit-rate 1m;
  }
  sched-best-effort-1 {
    transmit-rate 8m;
  }
  sched-bronze-1 {
    transmit-rate 6m;
  }
  sched-silver-1 {
    transmit-rate 4m;
  }
  sched-gold-1 {
    transmit-rate 2m;
  }
}

```

### Associating a Scheduler with a Fabric Priority

On M320 and T-series platforms only, you can associate a scheduler with a class of traffic that has a specific priority while transiting the fabric. Traffic transiting the fabric can have two priority values: low or high. To associate a scheduler with a fabric priority, include the priority and scheduler statements at the [edit class-of-service fabric scheduler-map] hierarchy level:

```

[edit class-of-service fabric scheduler-map]
priority (high | low) scheduler scheduler-name;

```



**NOTE:** For a scheduler that you associate with a fabric priority, you can include only the drop-profile-map statement at the [edit class-of-service schedulers *scheduler-name*] hierarchy level. You cannot include the buffer-size, transmit-rate, and priority statements at that hierarchy level.

---

For information about associating a forwarding class with a fabric priority, see “Overriding Fabric Priority Queuing” on page 823.

**Example: Associating a Scheduler with a Fabric Priority**

Associate a scheduler with a class of traffic that has a specific priority while transiting the fabric:

```
[edit class-of-service]
schedulers {
  fab-be-scheduler {
    drop-profile-map loss-priority low protocol any drop-profile fab-profile-1;
    drop-profile-map loss-priority high protocol any drop-profile fab-profile-2;
  }
  fab-ef-scheduler {
    drop-profile-map loss-priority low protocol any drop-profile fab-profile-3;
    drop-profile-map loss-priority high protocol any drop-profile fab-profile-4;
  }
}
drop-profiles {
  fab-profile-1 {
    fill-level 100 drop-probability 100;
    fill-level 85 drop-probability 50;
  }
  fab-profile-2 {
    fill-level 100 drop-probability 100;
    fill-level 95 drop-probability 50;
  }
  fab-profile-3 {
    fill-level 75 drop-probability 100;
    fill-level 95 drop-probability 50;
  }
  fab-profile-4 {
    fill-level 100 drop-probability 100;
    fill-level 80 drop-probability 50;
  }
}
fabric {
  scheduler-map {
    priority low scheduler fab-be-scheduler;
    priority high scheduler fab-ef-scheduler;
  }
}
```

## Configuring RED Drop Profiles

RED drop profiles are associated with the forwarding classes and loss priorities from the scheduler-map you configured on the interface. To configure the drop profiles themselves, include the drop-profiles statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
drop-profiles {
  profile-name {
    fill-level percentage drop-probability percentage;
    interpolate {
      fill-level [ values ];
      drop-probability [ values ];
    }
  }
}
```

In this configuration, you include either the interpolate statement and its options, or the fill-level and drop-probability *percentage* values. These two alternatives enable you to configure either each drop probability at up to 64 fill-level/drop-probability paired values, or a profile represented as a series of line segments.

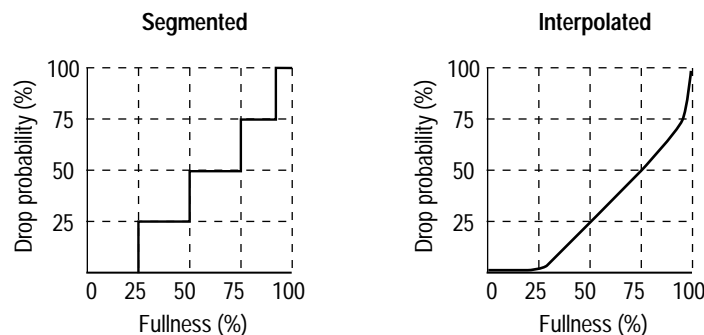


**NOTE:** If you configure the interpolate statement, you can specify more than 64 pairs, but the system generates only 64 discrete entries.

The line segments are defined in terms of the following graphical model: in the first quadrant, the x axis represents the fill level and the y axis represents the drop probability. The initial line segment spans from the origin (0,0) to the point ( $\langle l1 \rangle$ ,  $\langle p1 \rangle$ ); a second line runs from ( $\langle l1 \rangle$ ,  $\langle p1 \rangle$ ) to ( $\langle l2 \rangle$ ,  $\langle p2 \rangle$ ) and so forth, until a final line segment connects (100, 100). The system automatically constructs a drop profile containing 64 fill levels at drop probabilities that approximate the calculated line segments.

Figure 37 shows sample line graphs comparing use of the segment percentages (on the left) and interpolated values (on the right):

**Figure 37: Segmented and Interpolated Drop Profiles**



1704

## Packet Loss Priority

The system supports two packet loss priority (PLP) designations, low and high.

The packet loss priority is used to determine the RED drop profile when queuing a packet. You can set it by configuring a classifier or policer.

## Rewriting Packet Header Information

You can rewrite the packet header bits because the logical interface transmits the packet along with the forwarding-class and PLP information associated with the packet. If the packet's header field is blank, the forwarding class and PLP information specified in the rule are entered, or if the forwarding class and PLP information are already present, they are replaced with the new values. The traffic is then classified accordingly. The rewrite-rules configurations define the mappings.

Table 54 shows the default mappings.

**Table 54: Default Packet Header Rewrite Mappings**

Map from Forwarding Class	Map to DSCP/DSCP IPv6/ EXP/IEEE/IP	PLP Value
expedited-forwarding	ef	low
expedited-forwarding	ef	high
assured-forwarding	af11	low
assured-forwarding	af12 (DSCP/DSCP IPv6/EXP)	high
best-effort	be	low
best-effort	be	high
network-control	nc1/cs6	low
network-control	nc2/cs7	high

For the default bit definitions of DSCP, DSCP IPv6, EXP, and IEEE code points, see “Defining Code-Point Aliases” on page 818.

To configure a rewrite-rules mapping and associate it with the appropriate forwarding class and code-point alias or bit set, include the rewrite-rules statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
rewrite-rules {
  (dscp | dscp-ipv6 | exp | ieee-802.1 | inet-precedence) rewrite-name {
    import (rewrite-name | default);
    forwarding-class class-name {
      loss-priority level code-point (alias | bits);
    }
  }
}
```

The rewrite rule sets the code-point aliases and bit patterns for a specific forwarding class and PLP. The inputs for the map are the forwarding class and the PLP. The output of the map is the code-point alias or bit pattern. For more information about how CoS maps work, see Table 45 on page 804.

To assign the rewrite-rules configuration to the output logical interface, include the `rewrite-rules` statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy level:

```
[edit class-of-service interfaces interface-name unit logical-unit-number]
rewrite-rules {
  dscp (rewrite-name | default);
  dscp-ipv6 (rewrite-name | default);
  exp (rewrite-name | default) protocol protocol-types;
  exp-push-push-push default;
  exp-swap-push-push default;
  ieee-802.1 (rewrite-name | default);
  inet-precedence (rewrite-name | default);
}
```

You can use interface wildcards for *interface-name* and *logical-unit-number*.

You can also include Layer 2 and Layer 3 rewrite information in the same configuration.



**NOTE:** DSCP and DSCP IPv6 rewrite rules are not supported on IQ PICs installed on T-series platforms.

---

The following sections explain several rewrite rule applications you can configure:

Rewriting EXP Bits on a Particular Node on page 855

Rewriting MPLS and IPv4 Packet Headers on page 857

Rewriting the EXP Bits of All Three Labels of an Outgoing Packet on page 859

Rewriting IEEE 802.1p Packet Headers with MPLS EXP Value on page 861

### **Rewriting EXP Bits on a Particular Node**

To configure a custom table to rewrite the EXP bits, also known as CoS bits, on a particular node, the classifier table and the rewrite table must specify exactly the same code points.

In addition, the least significant bit of the code point itself must represent the PLP value. For example, code point 000 must be associated with PLP low, 001 must be associated with PLP high, and so forth.

**Example: Rewriting EXP Bits on a Particular Node**

Configure a custom table to rewrite the EXP bits on a particular node:

```
[edit class-of-service]
classifiers {
  exp exp-class {
    forwarding-class be {
      loss-priority low code-points 000;
      loss-priority high code-points 001;
    }
    forwarding-class af {
      loss-priority low code-points 010;
      loss-priority high code-points 011;
    }
    forwarding-class ef {
      loss-priority low code-points 100;
      loss-priority high code-points 101;
    }
    forwarding-class nc {
      loss-priority low code-points 110;
      loss-priority high code-points 111;
    }
  }
}
rewrite-rules {
  exp exp-rw {
    forwarding-class be {
      loss-priority low code-point 000;
      loss-priority high code-point 001;
    }
    forwarding-class af {
      loss-priority low code-point 010;
      loss-priority high code-point 011;
    }
    forwarding-class ef {
      loss-priority low code-point 100;
      loss-priority high code-point 101;
    }
    forwarding-class nc {
      loss-priority low code-point 110;
      loss-priority high code-point 111;
    }
  }
}
```

## Rewriting MPLS and IPv4 Packet Headers

You can apply a rewrite rule to MPLS and IPv4 packet headers simultaneously. This allows you to initialize MPLS EXP and IP precedence bits at LSP ingress. You can configure different rewrite rules depending on whether the traffic is VPN or non-VPN.

The default MPLS EXP rewrite table contents are shown in Table 55.

**Table 55: Default MPLS EXP Rewrite Table**

Forwarding Class	Loss Priority	Code Point
best-effort	low	000
best-effort	high	001
expedited-forwarding	low	010
expedited-forwarding	high	011
assured-forwarding	low	100
assured-forwarding	high	101
network-control	low	110
network-control	high	111

To override the default MPLS EXP rewrite table and rewrite MPLS and IPv4 packet headers simultaneously, include the protocol statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number* rewrite-rules exp *rewrite-rule-name*] hierarchy level:

```
[edit class-of-service interfaces interface-name unit logical-unit-number
rewrite-rules exp rewrite-rule-name]
protocol types;
```

The protocol statement defines the types of MPLS packets and packet headers to which the specified rewrite rule is applied. The MPLS packet can be a standard MPLS packet or an MPLS packet with an IPv4 payload. Specify the type of MPLS packet using the following options:

**mpls-any**—Applies the rewrite rule to MPLS packets and writes the code point value to MPLS headers.

**mpls-inet-both**—Applies the rewrite rule to VPN MPLS packets with IPv4 payloads. On M320 and T-series platforms, writes the code point value to the MPLS and IPv4 headers. On M-series routing platforms, causes all ingress MPLS LSP packets with IPv4 payloads to be initialized with 000 code points for IP precedence and MPLS EXP values.

**mpls-inet-both-non-vpn**—Applies the rewrite rule to non-VPN MPLS packets with IPv4 payloads. On M320 and T-series platforms, writes the code point value to the MPLS and IPv4 headers. On M-series routing platforms, causes all ingress MPLS LSP packets with IPv4 payloads to be initialized with 000 code points for IP precedence and MPLS EXP values.

An alternative to overwriting the default with a rewrite-rules mapping is to configure the default packet header rewrite mappings, as shown in Table 54 on page 854.

**Example: Rewriting MPLS and IPv4 Packet Headers**

On a M320 and T-series platform, configure rewrite tables and apply them in various ways to achieve the following results:

For interface so-3/1/0, the three EXP rewrite tables are applied to packets, depending on the protocol of the payload:

IPv4 packets (VPN) that enter the LSPs on interface so-3/1/0 are initialized with values from rewrite table exp-inet-table. An identical three-bit value is written into the IP precedence and MPLS EXP bit fields.

IPv4 packets (non-VPN) that enter the LSPs on interface so-3/1/0 are initialized with values from rewrite table rule-non-vpn. An identical three-bit value is written into the IP precedence and MPLS EXP bit fields.

Non-IPv4 packets that enter the LSPs on interface so-3/1/0 are initialized with values from rewrite table rule1, and written into the MPLS EXP header field only. The statement exp rule1 has the same result as exp rule1 protocol mpls-any.

For interface so-3/1/0, IPv4 packets transmitted over a non-LSP layer are initialized with values from IP precedence rewrite table rule2.

For interface so-3/1/1, IPv4 packets that enter the LSPs are initialized with values from EXP rewrite table exp-inet-table. An identical 3-bit value is written into the IP precedence and MPLS EXP bit fields.

For interface so-3/1/1, MPLS packets other than IPv4 Layer 3 types are also initialized with values from table exp-inet-table. For VPN MPLS packets with IPv4 payloads, the code point value is written to MPLS and IPv4 headers. For VPN MPLS packets without IPv4 payloads, the code point value is written to MPLS headers only.

```
[edit class-of-service]
rewrite-rules {
  exp exp-inet-table {
    forwarding-class best-effort {
      loss-priority low code-point 000;
      loss-priority high code-point 001;
    }
    forwarding-class assured-forwarding {
      loss-priority low code-point 010;
      loss-priority high code-point 011;
    }
    forwarding-class expedited-forwarding {
      loss-priority low code-point 111;
      loss-priority high code-point 110;
    }
    forwarding-class network-control {
      loss-priority low code-point 100;
      loss-priority high code-point 101;
    }
  }
}
```

```

exp rule1 {
  ...
}
inet-precedence rule2 {
  ...
}
exp rule_non_vpn {
  ...
}

interfaces {
  so-3/1/0 {
    unit 0 {
      rewrite-rules {
        exp rule1;
        inet-precedence rule2;
        exp exp-inet-table protocol mpls-inet-both;# for all VPN traffic
        exp rule_non_vpn protocol mpls-inet-both-non-vpn;
                                                    # for all non-VPN traffic
      }
    }
  }
  so-3/1/1 {
    unit 0 {
      rewrite-rules {
        exp exp-inet-table protocol [mpls-any mpls-inet-both];
      }
    }
  }
}

```

### Rewriting the EXP Bits of All Three Labels of an Outgoing Packet

In interprovider, carrier-of-carrier, and complex traffic engineering scenarios, it is sometimes necessary to push three labels on the next hop, using a swap-push-push or triple-push operation.

By default, on M-series routing platforms, the top MPLS EXP label of an outgoing packet is not rewritten when you configure swap-push-push and triple-push operations. On M-series routing platforms, you can rewrite the EXP bits of all three labels of an outgoing packet, thereby maintaining CoS of an incoming MPLS or non-MPLS packet.

To do this on incoming MPLS packets, include the `exp-swap-push-push` default statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number* rewrite-rules] hierarchy level:

```

[edit class-of-service interfaces interface-name unit logical-unit-number
rewrite-rules]
exp-swap-push-push default;

```

To do this on incoming non-MPLS packets, include the `exp-push-push-push` default statement at the `[edit class-of-service interfaces interface-name unit logical-unit-number rewrite-rules]` hierarchy level:

```
[edit class-of-service interfaces interface-name unit logical-unit-number
rewrite-rules]
exp-push-push-push default;
```

These configurations apply the default MPLS EXP rewrite table, as shown in Table 55 on page 857. You can configure these operations and override the default MPLS EXP rewrite table with a custom table. For more information about writing and applying a custom rewrite table, see “Rewriting Packet Header Information” on page 854.

### Example: Rewriting the EXP Bits of All Three Labels of an Outgoing Packet

Configure a swap-push-push operation, and override the default rewrite table with a custom table:

```
[edit class-of-service]
forwarding-classes {
  queue 0 be;
  queue 1 ef;
  queue 2 af;
  queue 3 nc;
}
interfaces {
  so-1/1/3 {
    unit 0 {
      rewrite-rules {
        exp exp_rew; # Apply custom rewrite table
        exp-swap-push-push default;
      }
    }
  }
}
rewrite-rules {
  exp exp_rew {
    forwarding-class be {
      loss-priority low code-point 000;
      loss-priority high code-point 100;
    }
    forwarding-class ef {
      loss-priority low code-point 001;
      loss-priority high code-point 101;
    }
    forwarding-class af {
      loss-priority low code-point 010;
      loss-priority high code-point 110;
    }
    forwarding-class nc {
      loss-priority low code-point 011;
      loss-priority high code-point 111;
    }
  }
}
```

## Rewriting IEEE 802.1p Packet Headers with MPLS EXP Value

For Ethernet interfaces installed on a M320 and T-series platform with a peer connection to an M-series routing platform or a T-series platform, you can rewrite both MPLS EXP and IEEE 802.1p bits to a configured value. This allows you to pass the configured value to the Layer 2 VLAN path.

To rewrite both the MPLS EXP and IEEE 802.1p bits, you must include EXP and IEEE 802.1p rewrite rules in the interface configuration. To configure EXP and IEEE 802.1p rewrite rules, include the `rewrite-rules` statement at the [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy level, specifying the `exp` and `ieee-802.1` options:

```
[edit class-of-service interfaces interface-name unit logical-unit-number]
rewrite-rules {
  exp rewrite-rule-name;
  ieee-802.1 default;
}
```

When you combine these two rewrite rules, only the EXP rewrite table is used for rewriting packet headers. If you do not configure a VLAN on the interface, only the EXP rewriting is in effect. If you do not configure an LSP on the interface or if the MPLS EXP rewrite rule mapping is removed, the IEEE 802.1p default rewrite rules mapping takes effect.



**NOTE:** You can also combine other rewrite rules. IP, DSCP, DSCP IPv6, and MPLS EXP are associated with Layer 3 packet headers, and IEEE 802.1p is associated with Layer 2 packet headers.

If you combine IEEE 802.1p with IP rewrite rules, the Layer 3 packets and Layer 2 headers are rewritten with the IP rewrite rule.

If you combine IEEE 802.1p with DSCP or DSCP IPv6 rewrite rules, three bits of the Layer 2 header and six bits of the Layer 3 packet header are rewritten with the DSCP or DSCP IPv6 rewrite rule.

---

The following example shows how to configure an EXP rewrite rule and apply it to both MPLS EXP and IEEE 802.1p bits:

```
[edit class-of-service]
rewrite-rules {
  exp exp-ieee-table {
    forwarding-class best-effort {
      loss-priority low code-point 000;
      loss-priority high code-point 001;
    }
    forwarding-class assured-forwarding {
      loss-priority low code-point 010;
      loss-priority high code-point 011;
    }
    forwarding-class expedited-forwarding {
      loss-priority low code-point 111;
      loss-priority high code-point 110;
    }
    forwarding-class network-control {
      loss-priority low code-point 100;
      loss-priority high code-point 101;
    }
  }
}
interfaces {
  so-3/1/0 {
    unit 0 {
      rewrite-rules {
        exp exp-ieee-table;
        ieee-802.1 default;
      }
    }
  }
}
```

## Configuring CoS-Based Forwarding

---

CoS-based forwarding (CBF) enables you to control next-hop selection based on a packet's class of service and, in particular, the value of the IP packet's precedence bits.

For example, you might want to specify a particular interface or next hop to carry high-priority traffic while all best-effort traffic takes some other path. When a routing protocol discovers equal cost paths, it can pick a path at random or load-share across the paths either through hash selection or round robin. CBF allows path selection based on class.

You can apply CBF only to a defined set of routes. Therefore you must configure a policy statement as in the following example:

```
[edit]
policy-options {
  policy-statement my-cos-forwarding {
    from {
      route-filter filter-name;
    }
    then {
      cos-next-hop-map map-name;
    }
  }
}
```

This configuration specifies that routes matching the route filter will be subject to the CoS next-hop mapping specified by *map-name*. For more information about configuring policy statements, see the *JUNOS Policy Framework Configuration Guide*.

To specify a CoS next-hop map, include the forwarding-policy statement at the [edit class-of-service] hierarchy level:

```
[edit class-of-service]
forwarding-policy {
  next-hop-map map-name {
    forwarding-class class-name {
      next-hop [ next-hop-name ];
      lsp-next-hop [ lsp-regular-expression ];
    }
  }
}
```

When you configure CBF with OSPF as the IGP, you must specify the next hop as an interface name or next-hop alias, not as an IP address. This is true because OSPF adds routes with the interface as the next hop for point-to-point interfaces; the next hop does not contain the IP address. For an example configuration, see “Example: Configuring CoS-Based Forwarding” on page 865.

The JUNOS software applies the CoS next-hop map to the set of next hops previously defined; the next hops themselves can be located across any outgoing interfaces on the routing platform. For example, the following configuration associates a set of forwarding classes and next-hop identifiers:

```
[edit class-of-service forwarding-policy]
next-hop-map map1 {
  forwarding-class expedited-forwarding {
    next-hop next-hop1;
    next-hop next-hop2;
  }
  forwarding-class best-effort {
    next-hop next-hop3;
    lsp-next-hop lsp-next-hop4;
  }
}
```

In this example, next-hop $N$  is either an IP address or an egress interface for some next hop, and lsp-next-hop4 is a regular expression corresponding to any next hop with that label. Q1 through Q $N$  are a set of forwarding classes that map to the specific next hop. That is, when a packet is switched with Q1 through Q $N$ , it will be forwarded out the interface associated with the associated next hop.

This configuration has the following implications:

A single forwarding class can map to multiple standard next hops or LSP next hops. This implies that load sharing is done across standard next hops or LSP next hops servicing the same class value. To make this work properly, the JUNOS software creates a list of the equal-cost next hops and forwards packets according to standard load-sharing rules for that forwarding class.

If a forwarding class configuration includes LSP next hops and standard next hops, the LSP next hops are preferred over the standard next hops. In the preceding example, if both next-hop3 and lsp-next-hop4 are valid next hops for a route to which map1 is applied, the forwarding table includes entry lsp-next-hop4 only.

If next-hop-map does not specify all possible forwarding classes, the default forwarding class is selected as the default. If the default forwarding class is not specified in the next-hop map, a default is designated randomly. The default forwarding class is the class associated with queue 0.

For LSP next hops, the JUNOS software uses UNIX regex(3)-style regular expressions. For example, if the following labels exist: lsp, lsp1, lsp2, lsp3, the statement lsp-next-hop lsp matches lsp, lsp1, lsp2, and lsp3. If you do not desire this behavior, you must use the anchor characters lsp-next-hop "^lsp\$", which match lsp only.

The final step is to apply the route filter to routes exported to the forwarding engine. This is shown in the following example:

```
routing-options {
  forwarding-table {
    export my-cos-forwarding;
  }
}
```

This configuration instructs the routing process to insert routes to the forwarding engine matching `my-cos-forwarding` with the associated next-hop CBF rules.

The following algorithm is used when you apply a configuration to a route:

If the route is a single next-hop route, all traffic will go to that route; that is, no CBF will take effect.

For each next hop, associate the proper forwarding class. If a next hop appears in the route but not in the `cos-next-hop` map, it will not appear in the forwarding table entry.

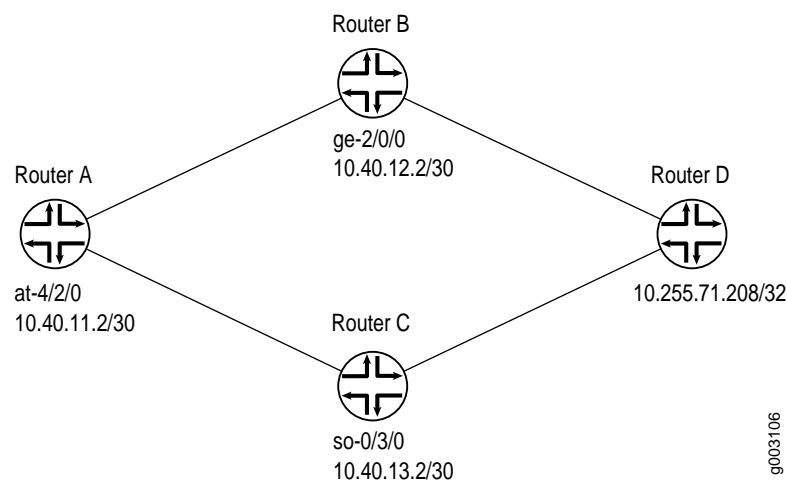
The default forwarding class is used if all forwarding classes are not specified in the next-hop map. If the default is not specified, one is chosen randomly.

### Example: Configuring CoS-Based Forwarding

Router A has two routes to destination 10.255.71.208 on Router D. One route goes through Router B, and the other goes through Router C, as shown in Figure 38.

Configure Router A with CBF to select Router B for queue 0 and queue 2, and Router C for queue 1 and queue 3.

Figure 38: Sample CoS-Based Forwarding



When you configure CBF with OSPF as the IGP, you must specify the next hop as an interface name, not as an IP address. The next hops in this example are specified as ge-2/0/0.0 and so-0/3/0.0.

```
[edit class-of-service]
forwarding-policy {
  next-hop-map my_cbf {
    forwarding-class be {
      next-hop ge-2/0/0.0;
    }
    forwarding-class ef {
      next-hop so-0/3/0.0;
    }
    forwarding-class af {
      next-hop ge-2/0/0.0;
    }
    forwarding-class nc {
      next-hop so-0/3/0.0;
    }
  }
}
classifiers {
  inet-precedence inet {
    forwarding-class be {
      loss-priority low code-points [ 000 100 ];
    }
    forwarding-class ef {
      loss-priority low code-points [ 001 101 ];
    }
    forwarding-class af {
      loss-priority low code-points [ 010 110 ];
    }
    forwarding-class nc {
      loss-priority low code-points [ 011 111 ];
    }
  }
}
forwarding-classes {
  queue 0 be;
  queue 1 ef;
  queue 2 af;
  queue 3 nc;
}
interfaces {
  at-4/2/0 {
    unit 0 {
      classifiers {
        inet-precedence inet;
      }
    }
  }
}
}
```

```

[edit policy-options]
policy-statement cbf {
  from {
    route-filter 10.255.71.208/32 exact;
  }
  then cos-next-hop-map my_cbf;
}

[edit routing-options]
graceful-restart;
forwarding-table {
  export cbf;
}

[edit interfaces]
traceoptions {
  file trace-intf size 5m world-readable;
  flag all;
}
so-0/3/0 {
  unit 0 {
    family inet {
      address 10.40.13.1/30;
    }
    family iso;
    family mpls;
  }
}
ge-2/0/0 {
  unit 0 {
    family inet {
      address 10.40.12.1/30;
    }
    family iso;
    family mpls;
  }
}
at-4/2/0 {
  atm-options {
    vpi 1 {
      maximum-vcs 1200;
    }
  }
  unit 0 {
    vci 1.100;
    family inet {
      address 10.40.11.2/30;
    }
    family iso;
    family mpls;
  }
}

```

**Configuring CoS-Based Forwarding for IPv6** Configure CBF next-hop maps and CBF LSP next-hop maps for IPv6 addresses. The following example shows a CBF next-hop map for IPv6 addresses.

You can configure a next-hop map with both IPv4 and IPv6 addresses, or you can configure separate next-hop maps for IPv4 and IPv6 addresses and include the `from family (inet | inet6)` statements at the `[edit policy-options policy-options policy-statement policy-name term term-name]` hierarchy level to ensure that only next-hop maps of a specified protocol are applied to a specified route.

If you do not configure separate next-hop maps and include the `from family (inet | inet6)` statements in the configuration, when a route uses two next hops (whether IPv4, IPv6, interface, or LSP next hop) in at least two of the specified forwarding classes, CBF is used for the route; otherwise, the CBF policy is ignored.

1. Define the CBF next-hop map:

```
[edit class-of-service]
forwarding-policy {
  next-hop-map cbf-map {
    forwarding-class best-effort {
      next-hop [ ::192.168.139.38 192.168.139.38 ];
    }
    forwarding-class expedited-forwarding {
      next-hop [ ::192.168.140.5 192.168.140.5 ];
    }
    forwarding-class assured-forwarding {
      next-hop [ ::192.168.145.5 192.168.145.5 ];
    }
    forwarding-class network-control {
      next-hop [ ::192.168.141.2 192.168.141.2 ];
    }
  }
}
```

2. Define the CBF forwarding policy:

```
[edit policy-options]
policy-statement ls {
  then cos-next-hop-map cbf-map;
}
```

3. Export the CBF forwarding policy:

```
[edit routing-options]
forwarding-table {
  export ls;
}
```

## Configuring Class of Service for Tunnels

For Adaptive Services, Link Services, and Tunnel PICs installed on T-series platforms and M-series platforms with enhanced FPCs, class-of-service information is preserved inside GRE and IP-IP tunnels.

For the ES PIC installed on T-series platforms and M-series platforms with enhanced FPCs, class-of-service information is preserved inside IPsec tunnels. For IPsec tunnels, you do not need to configure CoS, because the ES PIC copies the TOS byte from the inner IP header to the GRE or IP-IP header.

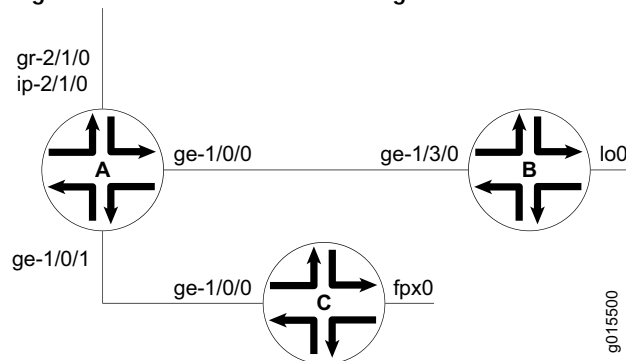
To configure CoS for GRE and IP-IP tunnels, perform the following configuration tasks:

1. To configure the tunnel, include the tunnel statement at the [edit interfaces *ip-fpc/pic/port* unit *logical-unit-number*] or [edit interfaces *gr-fpc/pic/port* unit *logical-unit-number*] hierarchy level.
2. To rewrite traffic on the outbound interface, include the rewrite-rules statement at the [edit class-of-service] and [edit class-of-service interfaces *interface-name* unit *logical-unit-number*] hierarchy levels. For GRE and IP-IP tunnels, you can configure IP precedence and DSCP rewrite rules.
3. To classify traffic on the inbound interface, you can configure a behavior aggregate (BA) classifier or firewall filter. Include the loss-priority and forwarding-class statements at the [edit firewall filter *filter-name* term *term-name* then] hierarchy level. Or you can include the classifiers statement at the [edit class-of-service] hierarchy level.

### Example: Configuring Class of Service for Tunnels

In Figure 39, Router A acts as a tunnel ingress device. The link between interfaces ge-1/0/0 in Router A and ge-1/3/0 in Router B is the GRE or IP-IP tunnel. Router A monitors the traffic received from interface ge-1/3/0. By way of interface ge-1/0/0, Router C generates traffic to Router B.

Figure 39: CoS with a Tunnel Configuration



```

Router A [edit interfaces]
ge-1/0/0 {
  unit 0 {
    family inet {
      address 10.80.0.2/24;
    }
  }
}
ge-1/0/1 {
  unit 0 {
    family inet {
      filter {
        input zf-catch-all;
      }
      address 10.90.0.2/24;
    }
  }
}
gr-2/1/0 {
  unit 0 {
    tunnel {
      source 11.11.11.11;
      destination 10.255.245.46;
    }
    family inet {
      address 21.21.21.21/24;
    }
  }
}
ip-2/1/0 {
  unit 0 {
    tunnel {
      source 12.12.12.12;
      destination 10.255.245.46;
    }
    family inet {
      address 22.22.22.22/24;
    }
  }
}

[edit routing-options]
static {
  route 1.1.1.1/32 next-hop gr-2/1/0.0;
  route 2.2.2.2/32 next-hop ip-2/1/0.0;
}

[edit class-of-service]
interfaces {
  ge-1/0/0 {
    unit 0 {
      rewrite-rules {
        inet-precedence zf-tun-rw-ipprec-00;
      }
    }
  }
}

```

```

rewrite-rules {
inet-precedence zf-tun-rw-ipprec-00 {
    forwarding-class best-effort {
        loss-priority low code-point 000;
        loss-priority high code-point 001;
    }
    forwarding-class expedited-forwarding {
        loss-priority low code-point 010;
        loss-priority high code-point 011;
    }
    forwarding-class assured-forwarding {
        loss-priority low code-point 100;
        loss-priority high code-point 101;
    }
    forwarding-class network-control {
        loss-priority low code-point 110;
        loss-priority high code-point 111;
    }
}
}
dscp zf-tun-rw-dscp-00 {
    forwarding-class best-effort {
        loss-priority low code-point 000000;
        loss-priority high code-point 001001;
    }
    forwarding-class expedited-forwarding {
        loss-priority low code-point 010010;
        loss-priority high code-point 011011;
    }
    forwarding-class assured-forwarding {
        loss-priority low code-point 100100;
        loss-priority high code-point 101101;
    }
    forwarding-class network-control {
        loss-priority low code-point 110110;
        loss-priority high code-point 111111;
    }
}

[edit firewall]
filter zf-catch-all {
    term term1 {
        then {
            loss-priority high;
            forwarding-class network-control;
        }
    }
}

Router B [edit interfaces]
ge-1/3/0 {
    unit 0 {
        family inet {
            address 10.80.0.1/24;
        }
    }
}
}

```

```

lo0 {
  unit 0 {
    family inet {
      address 10.255.245.46/32;
    }
  }
}

Router C [edit interfaces]
ge-1/0/0 {
  unit 0 {
    family inet {
      address 10.90.0.1/24;
    }
  }
}

[edit routing-options]
static {
  route 1.1.1.1/32 next-hop 10.90.0.2;
  route 2.2.2.2/32 next-hop 10.90.0.2;
}

```

### Overriding the Input Classification

For IPv4 or IPv6 packets, you can override the incoming classification, assigning them to the same forwarding class based on their input interface, input precedence bits, or destination address. You do so by defining a policy class when configuring CoS properties and referencing this class when configuring a routing policy.

When you override the classification of incoming packets, any mappings you configured for associated precedence bits or incoming interfaces to output transmission queues are ignored. Also, if the packet loss priority (PLP) bit was set in the packet by the incoming interface, the PLP bit is cleared.

To override the input packet classification, do the following:

1. Define the policy class by including the class statement at the [edit class-of-service policy] hierarchy level:

```

[edit class-of-service]
forwarding-policy {
  class class-name {
    classification-override {
      forwarding-class class-name;
    }
  }
}

```

*class-name* is a name that identifies the class.

2. Associate the policy class with a routing policy by including it in a policy-statement statement at the [edit policy-options] hierarchy level. Specify the destination prefixes in the route-filter statement and the CoS policy class name in the then statement.

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    from {
      route-filter destination-prefix match-type <class class-name>;
    }
    then class class-name;
  }
}
```

3. Apply the policy by including the export statement at the [edit routing-option] hierarchy level:

```
[edit routing-options]
forwarding-table {
  export policy-name;
}
```

## Examples: Configuring Class of Service

---

The first example includes classifiers, rewrite markers, and schedulers to configure a class of service policy. The second example, on page 878, shows how to configure a CoS policy specifically for IPv6 packets.

1. Define a classifier that matches IP traffic arriving on the interface. The affected IP traffic has IP precedence bits with patterns matching those defined by aliases A or B. The loss priority of the matching packets is set to low, and the forwarding class is mapped to best effort (queue 0):

```
[edit]
class-of-service {
  classifiers {
    inet-precedence normal-traffic {
      forwarding-class best-effort {
        loss-priority low code-points [my1 my2];
      }
    }
  }
}
```

Following are the code-point alias and forwarding-class mappings referenced in the normal-traffic classifier:

```
[edit]
class-of-service {
  code-point-aliases {
    inet-precedence {
      my1 000;
      my2 001;
      ...
    }
  }
}
```

```
[edit]
```

```

class-of-service {
  forwarding-classes {
    queue 0 best-effort;
    queue 1 expedited-forwarding;
  }
}

```

2. Use rewrite markers to redefine the bit pattern of outgoing packets. Assign the new bit pattern based on specified forwarding classes, regardless of the loss priority of the packets:

```

[edit]
class-of-service {
  rewrite-rules {
    inet-precedence clear-prec {
      forwarding-class best-effort {
        loss-priority low code-point 000;
        loss-priority high code-point 000;
      }
      forwarding-class expedited-forwarding {
        loss-priority low code-point 100;
        loss-priority high code-point 100;
      }
    }
  }
}

```

3. Configure a scheduler map associating forwarding classes with schedulers and drop-profiles:

```

[edit]
class-of-service {
  scheduler-maps {
    one {
      forwarding-class expedited-forwarding scheduler special;
      forwarding-class best-effort scheduler normal;
    }
  }
}

```

Schedulers establish how to handle the traffic within the output queue for transmission onto the wire. Following is the scheduler referenced in scheduler map one:

```

[edit]
class-of-service {
  schedulers {
    special {
      transmit-rate percent 30;
      priority high;
    }
    normal {
      transmit-rate percent 70;
      priority low;
    }
  }
}

```

4. Apply the normal-traffic classifier to all SONET/SDH interfaces and all logical interfaces of SONET/SDH interfaces; apply the clear-prec rewrite marker to all Gigabit Ethernet interfaces and all logical interfaces of Gigabit Ethernet interfaces; and apply the one scheduler map to all interfaces:

```
[edit]
class-of-service {
  interfaces {
    so-0/0/0 {
      scheduler-map one;
      unit 0 {
        classifiers {
          inet-precedence normal-traffic;
        }
      }
    }
    so-0/0/1 {
      scheduler-map one;
      unit 1 {
        classifiers {
          inet-precedence normal-traffic;
        }
      }
    }
    ge-1/0/0 {
      scheduler-map one;
      unit 0 {
        rewrite-rules {
          inet-precedence clear-prec;
        }
      }
      unit 1 {
        rewrite-rules {
          inet-precedence clear-prec;
        }
      }
    }
    ge-1/0/1 {
      scheduler-map one;
      unit 0 {
        rewrite-rules {
          inet-precedence clear-prec;
        }
      }
      unit 1 {
        rewrite-rules {
          inet-precedence clear-prec;
        }
      }
    }
  }
}
```

Following is the complete configuration:

```
[edit class-of-service]
classifiers {
  inet-precedence normal-traffic {
    forwarding-class best-effort {
      loss-priority low code-points [my1 my2];
    }
  }
}
code-point-aliases {
  inet-precedence {
    my1 000;
    my2 001;
    cs1 010;
    cs2 011;
    cs3 100;
    cs4 101;
    cs5 111;
    cs6 111;
  }
}
drop-profiles {
  high-priority {
    fill-level 20 drop-probability 100;
  }
  low-priority {
    fill-level 90 drop-probability 95;
  }
  big-queue {
    fill-level 100 drop-probability 100;
  }
}
forwarding-classes {
  queue 0 best-effort;
  queue 1 expedited-forwarding;
}
interfaces {
  so-0/0/0 {
    scheduler-map one;
    unit 0 {
      classifiers {
        inet-precedence normal-traffic;
      }
    }
  }
  so-0/0/1 {
    scheduler-map one;
    unit 1 {
      classifiers {
        inet-precedence normal-traffic;
      }
    }
  }
}
```

```

ge-1/0/0 {
  scheduler-map one;
  unit 0 {
    rewrite-rules {
      inet-precedence clear-prec;
    }
  }
  unit 1 {
    rewrite-rules {
      inet-precedence clear-prec;
    }
  }
}
ge-1/0/1 {
  scheduler-map one;
  unit 0 {
    rewrite-rules {
      inet-precedence clear-prec;
    }
  }
  unit 1 {
    rewrite-rules {
      inet-precedence clear-prec;
    }
  }
}
}
rewrite-rules {
  inet-precedence clear-prec {
    forwarding-class best-effort {
      loss-priority low code-point 000;
      loss-priority high code-point 000;
    }
    forwarding-class expedited-forwarding {
      loss-priority low code-point 100;
      loss-priority high code-point 100;
    }
  }
}
}
scheduler-maps {
  one {
    forwarding-class expedited-forwarding scheduler special;
    forwarding-class best-effort scheduler normal;
  }
}
}
schedulers {
  special {
    transmit-rate percent 30;
    priority high;
  }
  normal {
    transmit-rate percent 70;
    priority low;
  }
}
}

```

**Configuring a CoS Policy for IPv6 Packets**

1. Define a new classifier of type DSCP IPv6.

```
[edit class-of-service]
classifiers {
  dscp-ipv6 core-dscp-map {
    forwarding-class best-effort {
      loss-priority low code-points 000000;
    }
    forwarding-class assured-forwarding {
      loss-priority low code-points 001010;
    }
    forwarding-class network-control {
      loss-priority low code-points 110000;
    }
  }
}
```

2. Define a new rewrite rule of type DSCP IPv6.

```
[edit class-of-service]
rewrite-rules {
  dscp-ipv6 core-dscp-rewrite {
    forwarding-class best-effort {
      loss-priority low code-point 000000;
    }
    forwarding-class assured-forwarding {
      loss-priority low code-point 001010;
    }
    forwarding-class network-control {
      loss-priority low code-point 110000;
    }
  }
}
```

3. Assign the classifier and rewrite rule to a logical interface.

```
[edit class-of-service]
interfaces so-2/0/0 {
  unit 0 {
    classifiers { # Both dscp and dscp-ipv6 classifiers on this interface.
      dscp default;
      dscp-ipv6 core-dscp-map;
    }
    rewrite-rules { # Both dscp and dscp-ipv6 rewrite rules on this interface.
      dscp default;
      dscp-ipv6 core-dscp-rewrite;
    }
  }
}
```