Application Note

# Juniper Networks J-series Services Routers Quality of Service (QoS)

## Implementing QoS in Mixed Scenarios

Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, California 94089
USA
408.745.2000
1.888 JUNIPER
**www.juniper.net**

Part Number: 350141-001 Nov 2008

# Table of Contents



# List of Figures

# Introduction

As packet networks have evolved to deliver data, voice and video, quality of service (QoS) has become increasingly important to ensure that applications are afforded the preferential treatment required to operate properly at all times. This application note will provide the reader the prerequisite knowledge to configure Juniper Networks J-series services routers to support QoS in a branch office environment.

# Scope

The purpose of this application note is to explain the J-series QoS processing model, available options, and how to configure QoS using typical branch office scenarios.

The reader should be familiar with QoS basics, as this paper does not provide a detailed explanation of either QoS history or the different implementation approaches that the industry has taken. This application note focuses on J-series QoS through the use of deployment examples and configurations.

# Design Considerations

The choice of whether to deploy QoS in an IP network is often considered when network congestion starts to negatively impact end users. Network congestion can be alleviated through a careful QoS design, but may also be improved by simply increasing bandwidth. The network engineer must balance the costs and benefits of both approaches. The final design may include both an end-to-end QoS design and additional bandwidth across certain network links.

## Hardware Requirements

- J-series routers (J2320, J2350, J4300, J4350, J6300, J6350)

## Software Requirements

- JUNOS® software version 9.0 or higher

# Description and Deployment Scenario

In general terms, quality of service (QoS) is the ability of a network to provide a defined level of service for particular traffic. The problem with such a broad definition is that it is very open. What constitutes service quality? What are the parameters that affect service in a packet-based network? How do you define particular traffic, or, more specifically, how do you determine which traffic is given what service?

The first question can be answered by understanding that the following metrics are considered to be standard measurements of service quality:

- End-to-end packet delay
- Delay jitter
- Available capacity
- Drop probability

Throughout this application note, the above metrics will be used to help explain QoS models and methods.

To provide desired service quality, packets associated with target service levels or guarantees must be treated uniformly across networks. To be able to assign a specific type of service to a packet, traffic must first be identified as belonging to certain classes, a procedure commonly referred to as classification. As such, this paper will begin by discussing packet classification in JUNOS software.

## Traffic Classification

Different QoS proposals have been standardized by the IETF. In particular, JUNOS software adheres to the differentiated services model (based on RFC 2475). Packets are normally classified at the edge of the network by marking the Differentiated Services code point (DSCP) field in the IP header, which essentially defines the class of service for that packet as it moves through the network.

As traffic is forwarded, access devices are usually the first devices to mark packets. On J-series routers, which often serve as access devices, IP packets can be grouped into service classes and marked by:

- Source/destination IP address
- Source/destination port
- Protocol
- Application
- Ingress/egress interface
- Ingress/egress interface group
- Any combination of the above

Normally, classification is driven by application and business requirements. For example, imagine a corporate network that uses Session Initiation Protocol (SIP) phones and video conferencing equipment in conjunction with a proprietary billing application that is bound to a specific TCP port. Assume that the network also transports internal business application traffic and is connected directly to the Internet using VPN tunnels to other corporate sites. In this model, traffic could be classified as follows:

- Application-based classification for SIP traffic, allowing all data and control traffic to be assigned to a particular DSCP class
- Protocol/port-based classification for the billing application
- Zone-based classification to differentiate site-to-site VPN traffic from Internet traffic
- All remaining traffic

As packets are classified in JUNOS software, they are assigned to a forwarding class which specifies both the transmit queue and the packet loss priority. Borrowing terminology from the differentiated services model, the per-hop behavior of a packet (meaning how each hop in the network will treat a given packet) is determined by its specific queue and loss priority. Later sections will explain how traffic classes are specified and associated with queues, and how different loss priorities affect the drop probability of a packet.

Recent versions of JUNOS software also allow the creation of virtual channels. Virtual channels are available on J-series routers only, and provide hierarchical scheduling and shaping by allowing users to define a set of queues (virtual channels), each with its own scheduling and shaping parameters. In the initial examples, virtual channels are not used. The explanation of their use is deferred to later sections of this application note.

## Packet Walkthrough

J-series processing is based on the Juniper Networks M-/T-series hardware-forwarding model, but adds some platform specific capabilities.

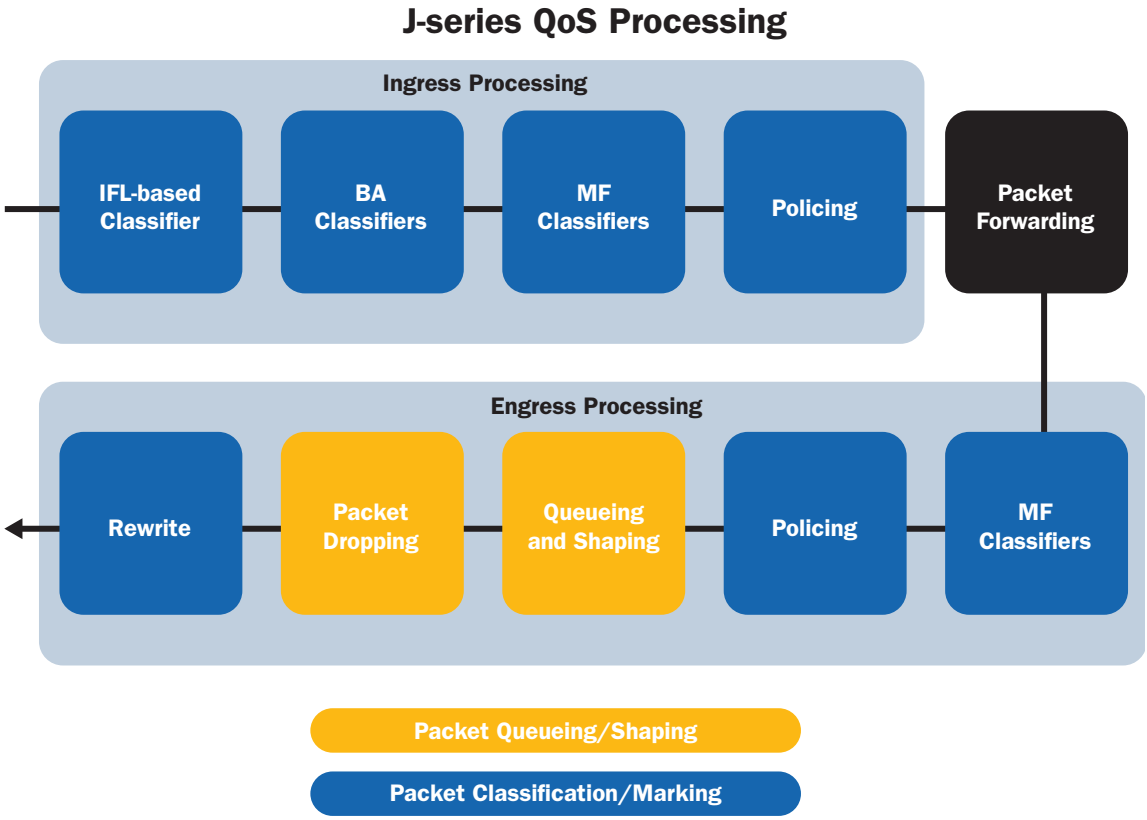


*Figure 1: QoS Processing Model*

Normally, packet classification or marking is performed at ingress as it is needed to ensure the correct treatment of the packets in the subsequent processing stages, while packet queuing/shaping is usually performed at egress since it is generally where bandwidth limitations occur, forcing packets to be either dropped or queued.

## Classification Stages

In the ingress pipeline, packets can be classified at multiple points, which allows for simple, computationally cheap classification as compared with more granular, computationally expensive classification. As expected, every time a packet is reclassified, the previous values are overwritten, so processing order is important. In order of application, the classification stages are:

### Logical Interface Classification

Logical interface classifiers allow for a simple and coarse marking where all traffic arriving on a given logical interface is assigned to the same forwarding class. The configuration simply associates a given logical interface with a forwarding class under the [class-of-service interface] hierarchy.

```
class-of-service {
    interfaces {
        <interface name>|* {
            unit <unit number> | * {
                forwarding-class <forwarding class name>;
            }
        }
    }
}
```

### BA Classification

Behavior aggregate (BA) classification allows for ingress grouping based on DSCP, 802.1p (for VLAN tagged Ethernet frames), and EXP (for MPLS packets) bits. Two standards, one based on RFC 791, and the other based on RFC 2474 which supersedes the former standard, are commonly followed when defining the per-hop behavior (PHB) of a packet using classifiers. However, both cannot be applied to the same logical interface, as they classify the packet based on the same field.

To apply a BA classifier, first define a classifier, a list of code points (values of the DSCP or EXP bit field), and the associated forwarding classes and packet loss priorities. Once the classifier is defined, it can be applied to a logical interface. The loss priority and forwarding class will be determined by querying the classifier table for that interface, using the ingress DSCP/EXP bits of the packet as the index value (the determination of which bits to query is based on the protocol family MPLS, IPv6, or IP).

For each protocol family, classifiers are configured under [class-of-service] as shown below:

```
class-of-service {
    classifiers {
        dscp|dscp-ipv6|exp|inet-precedence|ieee-802.1 {
            <classifier name> {
                            forwarding-class <forwarding class name> {
                  loss-priority <low|medium-low|medium-high|high> code-points [<list of
                  code point values or aliases>];
            }
            }
        }
    }
}

The newly configured classifier can then be applied to an interface:

class-of-service {
    interface  <interface name>|* {
         unit <unit number>|* {
            classifiers {
                dscp|dscp-ipv6|exp|inet-precedence <classifier name>;
            }
        }
    }
}
```

It is possible to assign more code points (combinations of the DSCP or EXP field) than forwarding classes/queues and drop priorities, which means that packets with different code points will receive the same PHB, which is perfectly normal.

Although not mandatory, to make configurations easy to read and reuse, code points can be given aliases that can be used when configuring a classifier.

```
class-of-service {
    code-point-aliases {
        dscp|dscp-ipv6|exp|inet-precedence {
            <alias name> <bit pattern>;
        }
    }
}
```

When no classifier is configured, JUNOS software uses the following default classifier, which maps all traffic to the best-effort forwarding class except for traffic with IP precedence 110 and 111, which is mapped to the network-control class.

**Table 1. Default Classifier**

| Code point | Forwarding Class | Loss priority |
|---|---|---|
| 000 | best-effort | low |
| 001 | best-effort | high |
| 010 | best-effort | low |
| 011 | best-effort | high |
| 100 | best-effort | low |
| 101 | best-effort | high |
| 110 | network-control | low |
| 111 | network-control | high |

### Multifield (MF) Classifiers

Classification can also be performed using stateless packet filters. Although a complete description of packet filters is outside the scope of this paper, packet filters can be used to modify both the forwarding class and loss priority.

This method of traffic classification is very granular, as it allows for the segregation of traffic based on the packet 5-tuple (source/destination IP, protocol, and source/destination ports). The configuration of an MF classifier requires the definition of a firewall filter as shown below, which is located under the [firewall family inet] hierarchy. Be aware that 802.1p bits cannot be used as matching conditions; by the time the packet is processed by the ingress/egress filters, the Ethernet frame has been stripped.

```
firewall {
    family inet {
        filter <filter name> {
            /* One or more terms. Terms are evaluated in order until either a match is
found or the last term is evaluated */
            term <term name> {
                from {
                    /* Matching conditions */
                }
                then {
                    forwarding-class <forwarding class name>;
                    loss-priority low|medium-low|medium-high|high;
                }
            }
        }
    }
}
```

### Simple Filters

Simple filters, which provide a subset of the functionality of complex filters, can also be defined. Simple filters are less computationally expensive than complex ones. In fact on some platforms other than the J-series, the packet forwarding engine (PFE) is not burdened with classification using simple filters. The configuration of a simple filter is identical to that of a standard firewall filter, but is located under the [firewall family inet simple-filter] hierarchy, with the following limitations:

- Allows only forwarding class, loss priority and policer actions (does not allow drop action; simple filters always accept traffic)
- Does not allow certain keywords/qualifiers such as except and protocol-except
- Does not allow non-contiguous masks
- Allows only one source-address and destination-address prefix per term

Filters and simple filters can be applied to logical interfaces both on ingress and egress:

```
interface <interface name> {
    unit <unit number> {
        family inet {
            filter|simple-filter {
                input <filter name>;
                output <filter name>;
            }
        }
    }
}
```

MF classifiers can also be applied independent of any ingress/egress interface as forwarding filters. When configured as a forwarding filter, the forwarding module applies filters regardless of the ingress or egress interface of the packet.

## Egress Classification

Egress classification applies MF classifiers to a logical interface in the outbound direction. Although less common than ingress classification, egress classification is sometimes used in situations where traffic from multiple interfaces is concentrated directly onto a single egress port and there is no LAN to LAN traffic (for instance a WAN link). In this case, a single egress MF classifier can be used to classify all of the egress traffic, instead of configuring multiple BA or MF classifiers on each ingress interface.

## Policing

Policing refers to the ability of a router to measure data rates and, based on this measurement, to either drop or reclassify the traffic.

J-series routers support single-rate policers that can be applied to traffic matching a particular ingress/egress filter. After MF classification is performed, it is possible to instruct a J-series router to measure the rate of the traffic matching the classifier, and either drop or change the forwarding class, or drop the priority of the packet if the measured rate exceeds a configurable threshold.

In simple terms, policers allow the establishment of a data rate, which, if exceeded, results in traffic being either reclassified or dropped. In order to measure traffic rates, it is important to determine a measurement interval (or burst limits, as we shall see shortly). Traffic always egresses an interface at line rate. To send traffic at a "lower speed," bursts have to be followed by idle periods, resulting in an average transmit rate lower than the line rate.

For example, to guarantee that no more than 100 Mbps egresses a given interface, we could choose to let up to 100 Mb be transmitted for a second, or 100 Kb for 1 ms, or any combination such that (traffic burst/interval) < desired bandwidth. The burst-size-limit parameter allows us to indirectly adjust the transmit interval. As a rule of thumb, allow a few milliseconds worth of traffic to be transmitted (burst size should minimally be equivalent to the maximum transmission unit size). For example, for a 5 ms burst, the burst size on a 1 Gbps interface should be burst size = data rate * interval = 1 Gbps * 5 ms = 5 Mb = 625 KB.

The configuration required uses firewall filters similar to those used for packet classification.

The first step is to define a policer under the [firewall policers] hierarchy, which allows the specification of a generic policer that can be applied to either single or multiple filters.

```
firewall {
    policer <policer name> {
        [logical-interface-policer];
        if-exceeding {
            bandwidth-limit <bandwidth in bps>;
            bandwidth-percent  <bandwidth as a percentage of the interface bw>;
            burst-size-limit <burst size in bytes>;
        }
        then {
            discard;
            forwarding-class <forwarding class name>;
            loss-priority [low|medium-low|medium-high|high];
        }
    }
}
```

Before explaining all of the configuration options, it is important to realize that policers can also be applied to a logical interface. In this case, all ingress/egress traffic (depending on the policer direction) is policed. When traffic matches more than one policer (for example, it first matches a filter-based policer and subsequently an interface policer), the order in which the policers are applied determines the overall result. In the case of stacked policers, every time traffic is non-conformant (exceeding the configured bandwidth limit), it will be either remarked or dropped.

A policer specifies a maximum bandwidth, a max-burst size, and an action. For interface-based policers, the maximum bandwidth can either be specified as an absolute value or as a percentage of the ingress/egress interface line rate. Filter-based policers can only specify the bandwidth in bps, as the transmit data rate it is not known until the egress interface is determined.

Once a policer is defined, it can be applied to a logical interface on either ingress or egress.

```
interface {
    <interface name> {
        unit <unit number> {
            family inet {
                policer input <policer name>;
                policer output <policer name>;
            }
        }
    }
}
```

Or, it can be specified as an action in a firewall filter:

```
firewall {
    family inet {
        filter <filter name> {
            term <term name> {
                from {
                    /* Match Conditions */
                }
                then {
                    policer <policer name>;
                    /* Other actions */
                }
            }
        }
    }
}
```

Using the next term action on a filter doing policing allows the stacking of one or more policers, whenever it is desired to limit aggregated traffic from multiple sources and/or interfaces from exceeding a particular bandwidth.

Multiple filters/terms can refer to the same policer, in which case the policer will calculate the aggregate bandwidth used by all of the traffic referring to that particular policer. The filter-specific keyword can be used to change this behavior, which results in new policers being defined every time the policer is referenced.

The policers we have seen to this point operate within a particular layer 3 protocol family (in particular IPv4). What is called a logical interface policer, on the other hand, can be applied at layer 2 of a logical interface to limit all ingress/egress traffic. The policer can be applied under the [interface unit] stanza.

```
interface {
    <interface name> {
        unit <unit number> {
            layer2-policer {
                input-policer <policer name>;
                output-policer <policer name>;
            }
        }
    }
}
```

## Queuing

After traffic is classified, queuing and scheduling can be used to provide different levels of service for the classified traffic. In JUNOS software, forwarding classes are associated with packet queues. A traffic class is no more than a queue that is configured with a given bandwidth and scheduling priority. Several forwarding classes can be assigned to the same queue (in that sense, different forwarding classes act as aliases for a particular queue).

```
class-of-service {
    forwarding-classes {
        /* The queue number is a value from 0 to 7 for a J-series router as it supports up
to 8 */
        queue <queue number> <forwarding class name> {
            priority low|high;
        }
    }
}
```

By default, every interface is configured with a set of four queues and four forwarding classes as follows:

**Table 2. Default Forwarding Classes**

| Forwarding Class | ID | Queue |
|---|---|---|
| best-effort | 0 | 0 |
| expedited-forwarding | 1 | 1 |
| assured-forwarding | 2 | 2 |
| network-control | 3 | 3 |

Unless specifically configured, J-series routers always use queue 3 (associated with the network-control forwarding class by default) to send high priority control packets.

### Defining Schedulers

Once forwarding classes (and therefore queues) have been defined, it is possible to configure the way each queue is scheduled. J-series routers implement a modified deficit round-robin (MDRR) scheduling algorithm and priority queuing. Each queue is configured with a given priority and transmit rate. The priority of a queue determines the order in which it is served, while the transmit rate defines how many bytes can be transmitted per scheduled interval.

The simplest configuration of schedulers consists of a priority and a transmit rate for each queue. The first step is to create a scheduler template with these parameters, and a scheduler map that associates each scheduler (transmit rate and priority) with a forwarding class (and therefore with a queue).

```
class-of-service {
    schedulers <name> {
        priority strict-high|high|medium-high|medium-low|low;
        transmit-rate (<rate> | percent <percentage> |  remainder) [exact];
    }
}

class-of-service {
    scheduler-maps {
        <name> {
            forwarding-class <forwarding class name> scheduler <scheduler name>;
        }
    }
}
```

Queues are scheduled from high to low priority. Since the scheduling algorithm enforces the configured transmit rate, starvation of low priority queues is prevented (a detailed discussion about the MDRR algorithm is outside the scope of this document). Queues with the same priority are served in a round robin fashion.

Treatment of the strict-high-priority queue, however, is different from other queues. Only one strict-high-priority queue can be configured on a logical interface. This queue has the highest priority and can use all of an interface's bandwidth (on a strict-high-priority queue it is not possible to configure a transmit rate). A strict-high-priority queue minimizes packet delays, as it is always serviced as soon as packets arrive but can starve other queues. To avoid unintentional starvation of other queues, it is a good idea to use policers in conjunction with strict-high-priority queuing.

Policers can be configured to mark packets as out of conformance, which alerts the scheduler to allow the packets to be forwarded only if the interface is not experiencing congestion. When an interface is experiencing congestion, out of conformance packets are automatically discarded. (A sample configuration is provided in the Stacked Policer with Out-of-Conformance Marking section of this document.)

Transmit rates can be either loose or exact. When the exact keyword is used, queues are not allowed to exceed the configured transmit rate even when other queues have no packets to send. Alternatively, it is possible to configure a shaping rate per queue, which specifies the maximum bandwidth a particular queue can use when the interface has bandwidth available. By default, a queue is allowed to use all of the available bandwidth (up to the interface's shaping rate), as long as other queues for the interface are idle. The addition of a shaping rate bounds how much traffic a queue can borrow.

```
class-of-service {
    schedulers <name> {
        shaping-rate (<rate> | percent <percentage>);
    }
}
```

### Applying Schedulers

Depending on how the interface is configured, J-series routers support up to eight queues per logical or physical interface. Each physical interface can be assigned queue sets shared between all logical interfaces associated with a physical interface, or each logical interface can be assigned its own queue set, which can be configured by adding the per-unit-scheduler option to the configuration.

```
interface {
    <interface name> {
        per-unit-scheduler | shared-scheduler;
    }
}
```

Shared scheduler interfaces support one set of queues and a scheduler that is used by all of its logical interfaces. In this case, the configuration of the queuing and scheduling algorithms consist of defining a scheduler map (as shown in the Defining Schedulers  section of this document) and applying it to the interface in the desired direction (generally egress).

```
class-of-service {
    interfaces <interface name> {
        scheduler-map <scheduler map name>;
    }
}
```

Shared scheduler interfaces have a maximum of eight queues shared between their logical interfaces. This allows the creation of different traffic classes, but not resource reservation for each logical interface. Since there will be contention between logical interfaces using the same shared resource, it is possible for a single logical interface to monopolize all available resources.

A per-unit scheduler interface supports a different scheduler configuration on each of its logical interfaces. The configuration of the scheduling algorithm for the set of queues of a logical interface is located under the scheduler-map [class-of-service interface] hierarchy.

```
class-of-service {
    interface <interface name> {
        unit <unit number | *> {
            scheduler-map <scheduler map name>;
        }
    }
}
```

As opposed to the single scheduler case, each logical interface configured with a scheduler map will have its own set of queues. It is possible to use a wildcard, which indicates that the default scheduler map is to be used for every logical interface configured on the physical interface.

### Configuration Process

The following diagram summarizes the process required to configure and apply a scheduler map to a physical or logical interface. The QoS configuration of most interfaces is usually the same, which is why scheduler maps are configuration templates that can be applied to multiple interfaces.

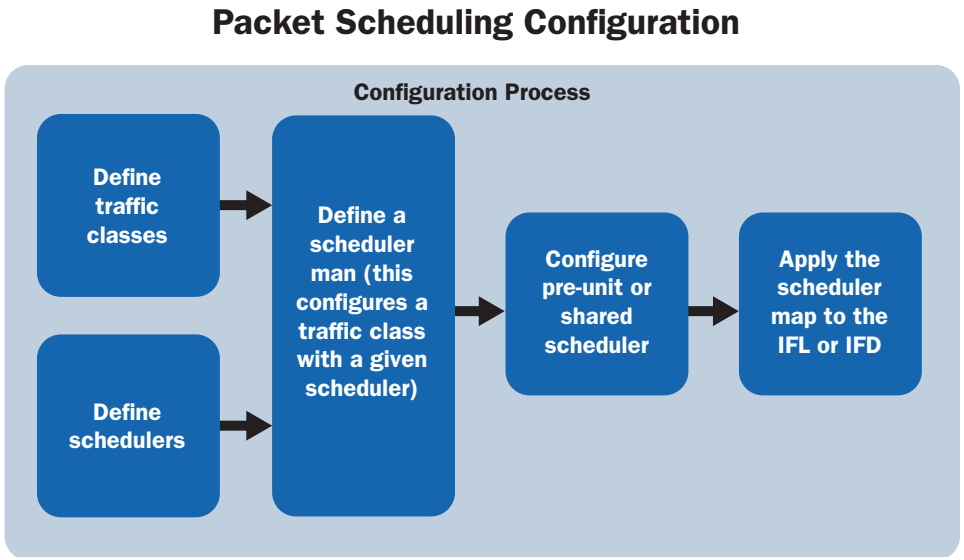


*Figure 2: Scheduling Configuration Summary*

## Traffic Dropping

J-series routers implement the weighted random early detection (WRED) algorithm, which was designed to prevent TCP synchronization when links experience congestion. When data networks become congested and drop packets, TCP sessions that suffer packet loss will reduce their window size to avoid congestion. Indiscriminate packet drops will (statistically speaking) signal the transmitting endpoints to slow their transmission rates, causing most of the senders to exponentially decrease their transmission rates simultaneously. This phenomenon is known as TCP synchronization, and it leads to bandwidth fluctuations on congested links. When synchronization occurs, senders reduce their transmission rates simultaneously, and slowly increase them again until links become congested—a process that then repeats itself.

The random early detection (RED) algorithm solves this by randomly dropping packets as queues become full. The drop probability can be configured as a function of queue size at any given time, so the more congestion, the more aggressive the drop profile. Randomly dropping traffic before an interface becomes congested signals end hosts to slow down, preventing an overloaded queue.

As discussed in the packet classification section, it is possible to assign packets to a forwarding class using either BA or MF classification. MF and BA classifiers can also be used to assign packets a loss priority. When packets are queued, the loss priority is used to determine which drop profile will be applied to a particular packet.

Loss profiles are configured under the [class-of-service drop-profiles] hierarchy and can be specified in two ways. By specifying a set of fill levels with associated drop probabilities, a staircase type of profile can be created for which the drop probability is assumed to be constant between fill levels.

```
class-of-service {
    drop-profiles <name> {
        fill-level <percentage> drop-probability <percentage>;
    }
}
```

The following example generates a staircase profile as illustrated below. (Note that the drop probability for a full queue is 100 percent.)

```
class-of-service {
    drop-profile staircase-test {
        fill-level 50 drop-probability 5;
        fill-level 60 drop-probability 10;
        fill-level 70 drop-probability 15;
    }
}
```

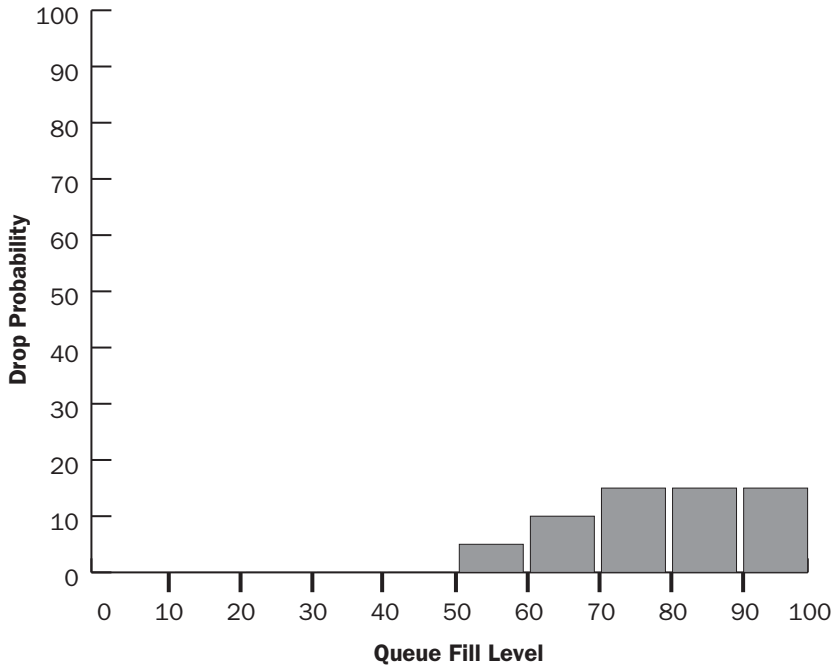


*Figure 3: Staircase Drop Profile*

Interpolated profiles create piecewise linear functions using the following configuration:

```
class-of-service {
    drop-profiles <name> {
        interpolate {
drop-probability [<values>] ;
fill-level [<values>];
        }
    }
}
```

As an example, the drop profile interpolate test shown below generates the profile illustrated in Figure 4 (note how the drop probability for a full queue is implicitly set to 100 percent):

```
class-of-service {
    drop-profile interpolate-test {
        interpolate {
            fill-level [50 60 70 80];
            drop-probability [0 5 10 15];
    }
    }
```

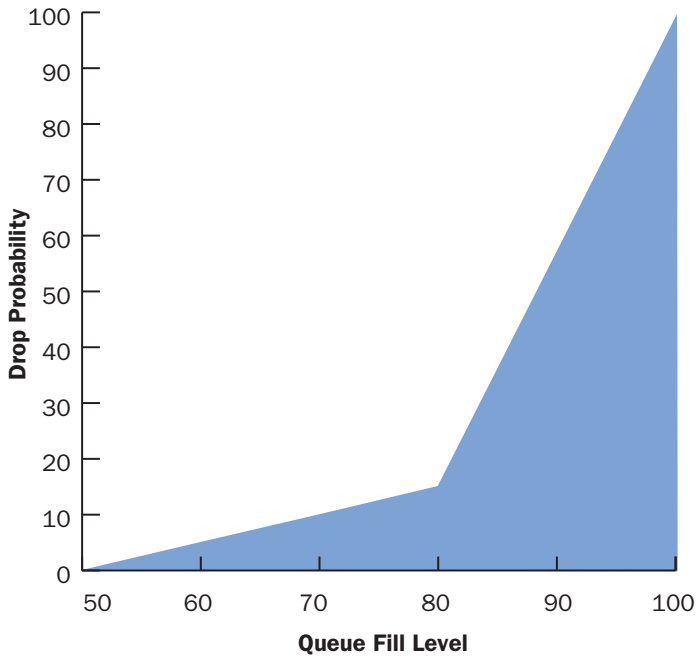


*Figure 4: Interpolated Profile*

Drop profiles can be configured for each forwarding class (queue) and drop priority. Packets with different drop priorities can have different drop profiles. When the RED algorithm is weighted by drop priority, it is normally referred to as weighted RED (WRED).

Once drop profiles are configured, they can be assigned to a scheduler.

```
class-of-service {
    scheduler <name> {
        drop-profile-map loss-priority low|medium-low|medium-high|high|any protocol any
drop-profile <name>;
    }
}
```

## Rewrite Rules

Rewrite rules change the marking of packets based on the forwarding class and loss priority combination as they egress the router. By default, J-series routers will not change the DSCP/precedence fields of forwarded packets.

Depending on the protocol, the DSCP, IP Precedence, MPLS EXP, 802.1p, DSCP for IPv6 traffic, and Frame Relay discard eligible (DE) bits can be modified. It is also possible to apply more than one classifier to the same egress queue/drop priority combination whenever the egress packet stacks more than one protocol. For example, packets exiting a VLAN tagged interface can have both their DSCP and 802.1p bits changed simultaneously. Not every packet encapsulation allows all possible rewrites. For example, the 802.1p bits can be changed only when the egress packet is a VLAN tagged packet, and the Frame Relay DE bit can only be set (or unset) for Frame Relay packets.

Configuration consists of defining the bit values to be written (or alias name if an alias has been defined) for each particular forwarding class and drop priority combination.

```
class-of-service {
    rewrite-rules {
        dscp|dscp-ipv6|exp|frame-relay-de|ieee-802.1p|inet-precedence <rule name> {
            /* One or more of the following can be configured */
            forwarding-class <class name> loss-priority high|medium-high|medium-low|low
code-point <bit string>|<code point alias>;
}
    }
}
```

After a rewrite rule is defined, it can be applied to a scheduler, which will then be applied to a queue via the application of a scheduler map (as shown in the Defining Schedulers section of this document).

## Configuration Examples

In this section, configuration examples are provided to help clarify earlier concepts.

### Logical Interface Classifier

This first example illustrates the use of a logical interface classifier in the simple network illustrated below.

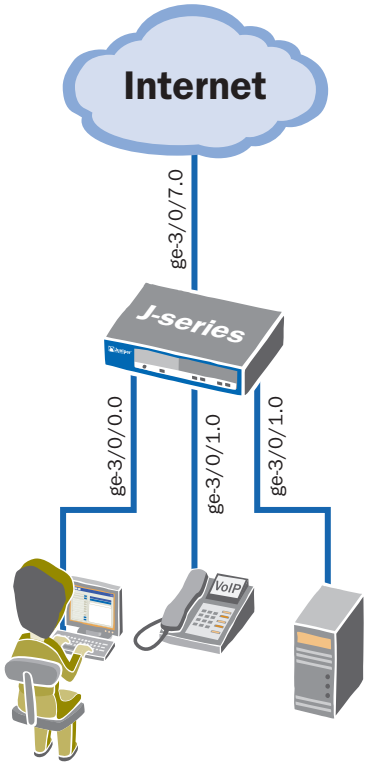


*Figure 5: Logical Interface Classifier*

In this network, traffic will be classified at the ingress interface. We will use the J-series default forwarding classes and assign VoIP traffic to the expedited forwarding class, high priority traffic to the assured forwarding class, and all other traffic to the best-effort class.

```
class-of-service {
   interfaces {
      ge-3/0/0 {
         unit 0 {
           forwarding-class best-effort;
         }
      }
      ge-3/0/1 {
         unit 0 {
            forwarding-class expedited-forwarding;
         }
      }
      ge-3/0/2 {
         unit 0 {
             forwarding-class assured-forwarding;
         }
      }
   }
}
```

### BA Classifier

We've seen in the previous example how to classify ingress traffic on an interface, but what if traffic priorities cannot be determined at the ingress interface? In this example, a J-series router sits between the Internet and a switching network, so traffic from different devices is received on interface ge-3/0/0.0 but is tagged differently.

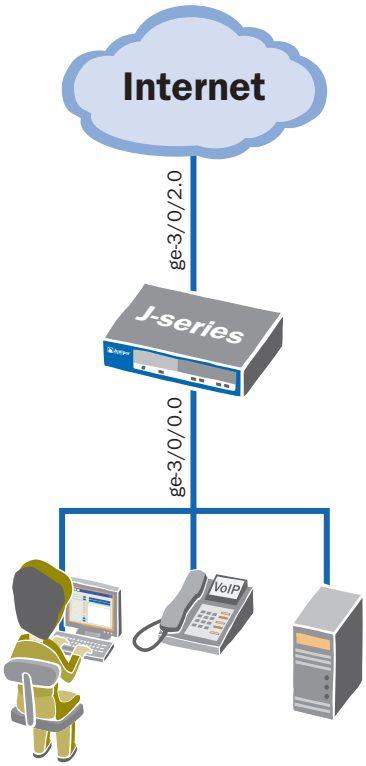


*Figure 6: BA Classification*

The J-series router receives three types of traffic on the ge-3/0/0.0 logical interface. VoIP phones will mark packets with a DSCP value of 101110; the server will mark high priority traffic with a DSCP value of 001010; and all other traffic regardless of marking will be treated as best-effort.

As in the previous example, we will be using JUNOS software default forwarding classes, assigning VoIP traffic to the expedited forwarding class, high priority traffic to the assured forwarding class, and all other traffic to the best-effort class.

```
class-of-service {
   classifiers {
      dscp custom {
         forwarding-class expedited-forwarding {
            loss-priority low code-points voip;
         }
         forwarding-class assured-forwarding {
            loss-priority low code-points high-priority;
         }
      }
   }
   code-point-aliases {
      dscp {
         voip 101110;
         high-priority 001010;
      }
   }
```

```
   interfaces {
      ge-3/0/6 {
         unit 0 {
            classifiers {
               dscp custom;
            }
         }
      }
   }
}
```

It is not necessary to specify any classification for best-effort traffic class because non-classified traffic is assigned to this class by default.

### MF Classifier

In the previous example, we described how to classify packets based on DSCP bits, but what if the packets arrive unmarked? We would still like to be able to classify them, which we can do by using the packet 5-tuple. This method and the previous one are not mutually exclusive. You can use BA classification for marked traffic and you can use MF classification for the balance (MF classification will override any previous BA classification).

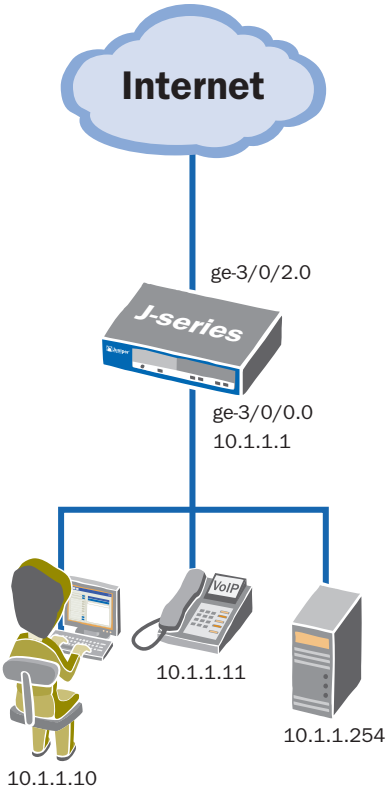


*Figure 7: MF Classification*

```
firewall {
   family inet {
      filter MF-classifier {
         term voip {
            from {
               source-address {
                  10.1.1.11/32;
```

```
                }
            }
            then {
               forwarding-class expedited-forwarding;
               accept;
            }
         }
         term server {
            from {
               source-address {
                  10.1.1.254/32;
               }
            }
            then {
               forwarding-class assured-forwarding;
               accept;
            }
         }
      term any {
         then {
            forwarding-class best-effort;
            accept;
         }
      }
   }
}
interfaces {
   ge-3/0/0 {
      unit 0 {
         family inet {
            filter {
               input MF-classifier;
            }
            address 10.1.1.1/24;
         }
      }
   }
}
```

### Interface Policer

Referring to our first example (logical interface classifier), we can rate-limit traffic on each logical interface so that no traffic class uses all available bandwidth. Let's use the following limits for our example network assuming a 10 Mbps uplink.

**Table 3. Interface Rate Limiting Example**

| Interface | Bandwidth |
|---|---|
| ge-3/0/0.0 | 1 Mbps Shared |
| ge-3/0/1.0 | 1 Mbps Shared |
| ge-3/0/2.0 | 1 Mbps |
| ge-3/0/3.0 | 8 Mbps |

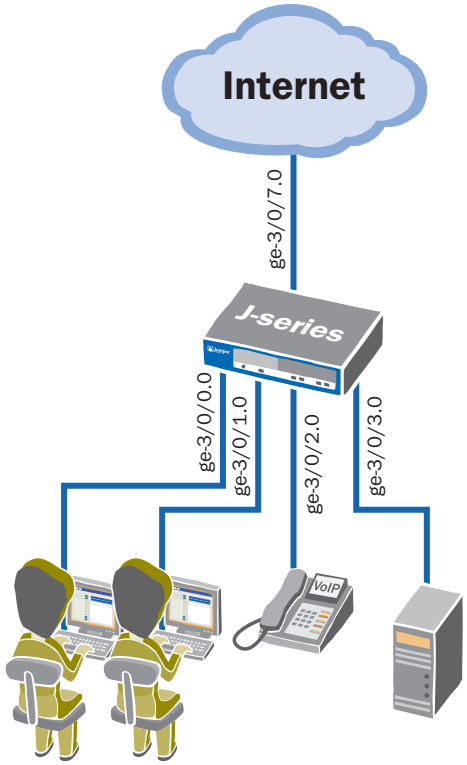


*Figure 8: Interface-Based Policing*

The resulting configuration is:

```
interface {
   ge-3/0/0 {
       unit 0 {
         family inet {
            policer {
               input be;
            }
         }
      }
   }
   ge-3/0/1 {
      unit 0 {
         family inet {
            policer {
               input be;
            }
         }
      }
   }
   ge-3/0/2 {
      unit 0 {
         family inet {
            policer {
               input voice;
            }
         }
      }
   }
```

```
   ge-3/0/3 {
      unit 0 {
         family inet {
            policer {
               input server;
            }
         }
      }
   }
firewall {
   policer voice {
      if-exceeding {
         bandwidth-limit 1m;
         burst-size-limit 3k;
      }
      then discard;
   }
   policer server {
      if-exceeding {
         bandwidth-limit 8m;
         burst-size-limit 6k;
      }
      then discard;
   }
   policer be {
      if-exceeding {
         bandwidth-limit 1m;
         burst-size-limit 3k;
      }
      then discard;
}
```

### Filter-Based Policer

Using the MF classifier example where all traffic is untagged, we will now rate-limit both marked and unmarked ingress traffic on the same interface. The network is the same as the one used in that previous example and as shown in Figure 7: MF Classification.

Let's continue to assume that VoIP and best-effort traffic are to be limited to 1 Mbps, and high-priority server traffic is to be limited to 8 Mbps. The configuration of the policers is the same as in the previous example.

```
firewall {
   policer voice {
      if-exceeding {
         bandwidth-limit 1m;
         burst-size-limit 3k;
      }
      then discard;
   }
   policer server {
      if-exceeding {
         bandwidth-limit 8m;
         burst-size-limit 6k;
      }
      then discard;
   }
   policer be {
      if-exceeding {
         bandwidth-limit 1m;
```

```
         burst-size-limit 3k;
      }
      then discard;
   }
}
```

However, instead of applying policers to logical interfaces, we apply them using packet filters, which allows for both classification and rate-limiting of packets simultaneously.

```
firewall {
   family inet {
      filter MF-classifier-and-policing {
         term voip {
            from {
               source-address {
                  10.1.1.11/32;
               }
            }
            then {
               forwarding-class expedited-forwarding;
               policer voice;

               accept;
            }
         }
         term server {
            from {
               source-address {
                  10.1.1.254/32;
               }
            }
            then {
               forwarding-class assured-forwarding;
               policer high-priority;
               accept;
            }
         }
         term any {
            then {
               forwarding-class best-effort;
               policer be;
               accept;
            }
         }
      }
   }
interfaces {
   ge-3/0/0 {
      unit 0 {
         family inet {
            filter {
               input MF-classifier-and-policing;
            }
            address 10.1.1.1/24;
         }
      }
   }
}
```

### Stacked Policers

What if we want to rate-limit the traffic in the previous example, but also ensure that the combined bandwidth of the VoIP and best-effort traffic does not exceed 1.5 Mbps (instead of 2 Mbps, which would be the maximum rate if we used two 1 Mbps policers)?

Stacked policers can be used to achieve the desired effect. First, we define policers just as we did before, but we add a 1.5 Mbps policer combination. When specifying the input filter, the next term action will be used to configure an extra policer applied to either the VoIP or best-effort traffic.

```
firewall {
policer voip {
    if-exceeding {
        bandwidth-limit 1m;
        burst-size-limit 3k;
    }
    then discard;
}
policer server {
    if-exceeding {
        bandwidth-limit 8m;
        burst-size-limit 6k;
    }
    then discard;
}
policer be {
    if-exceeding {
        bandwidth-limit 1m;
        burst-size-limit 3k;
    }
    then discard;
}
policer voip+be {
    if-exceeding {
        bandwidth-limit 1500000;
        burst-size-limit 3k;
    }
    then discard;
}
family inet {
    filter MF-classifier-and-policing {
        term server {
            from {
                source-address {
                    10.1.1.254/32;
                }
            }
            then {
                policer server;
                forwarding-class assured-forwarding;
                accept;
            }
        }
        term voip {
            from {
                source-address {
                    10.1.1.11/32;
                }
            }
```

```
            then {
                policer voip;
                forwarding-class expedited-forwarding;
                next term;
            }
        }
        term be {
            from {
                source-address {
                    10.1.1.11/32 except;
                }
            }
            then {
                policer be;
                forwarding-class best-effort;
                next term;
            }
        }
        term voip+be {
            then {
                policer voip+be;
                accept;
            }
        }
    }
}
```

### Stacked Policer with Out-of-Conformance Marking

Stacked policers can be very useful when dealing with oversubscribed interfaces, or when changing the drop profile of the traffic according to the transmitted rate.

As previously discussed, a strict-high-priority queue assigns all arriving traffic the highest priority and allows for the full use of the interface. Accordingly, it is good practice to limit the amount of traffic sent to this queue. Additionally, when mapping traffic to this queue, it can be classified as non-conforming. In the event of congestion, non-conforming traffic is rapidly discarded.

Specifically, a stacked policer can be used to establish a rate, which, if exceeded, causes packets to be classified as non-conforming and subsequently dropped. Continuing with the previous examples, we want to keep VoIP traffic under 2 Mbps and classify anything between 1 and 2 Mbps as out of conformance.

```
firewall {
policer voip {
    if-exceeding {
        bandwidth-limit 1m;
        burst-size-limit 6k;
    }
    then out-of-profile;
}
policer server {
    if-exceeding {
        bandwidth-limit 10m;
        burst-size-limit 625k;
    }
    then discard;
}
policer be {
    if-exceeding {
        bandwidth-limit 1m;
```

```
        burst-size-limit 6k;
    }
    then discard;
}
policer voip-excess {
    if-exceeding {
        bandwidth-limit 2m;
        burst-size-limit 6k;
    }
    then discard;
}
family inet {
    filter MF-classifier-and-policing {
        term server {
            from {
                source-address {
                    10.1.1.254/32;
                }
            }
            then {
                policer server;
                forwarding-class assured-forwarding;
                accept;
            }
        }
        term voip {
            from {
                source-address {
                    10.1.1.11/32;
                }
            }
            then {
                policer voip;
                forwarding-class expedited-forwarding;
                next term;
            }
        }
        term voip-excess {
            from {
                source-address {
                    10.1.1.11/32;
                }
            }
            then policer voip-excess;
        }
        term be {
            then {
                policer be;
                forwarding-class best-effort;
                accept;
            }
        }
    }
}
}
```

For this configuration to be useful, we have yet to give the expedited-forwarding class a strict-high priority. Our next example will show how this is done.

### Queuing

Previous examples illustrated how to mark and rate-limit traffic, but did not explain how to configure queuing. Continuing with our sample network shown in Figure 7: MF Classification, we will add queuing to the egress interface connected to the Internet.

The system default traffic classes will be used, but specific scheduling will be configured. For this example, the egress interface has 10 Mbps available, and traffic shaping will be used to allocate available bandwidth. When all queues are in use, the scheduling algorithm will allocate bandwidth according to Table 4, but if a queue (or queues) is not active, the unused bandwidth will be proportionately allocated to the rest of the queues.

As voice traffic is highly delay-sensitive, the voice traffic class will be assigned strict-high priority, which means that rate limiting has to be used to limit the maximum transmit rate this queue is allowed to use.

**Table 4. Queuing Example**

| Forwarding Class | Bandwidth | Priority |
|---|---|---|
| **best-effort** | 500 Kbps | low |
| **expedited-forwarding** | 1 Mbps (rate limited) | strict-high |
| **assured-forwarding** | 8 Mbps | medium-high |
| **network-control** | 500 Kbps | high |

```
firewall {
    policer voice {
        if-exceeding {
            bandwidth-limit 1m;
            burst-size-limit 6k;
        }
        then discard;
    }
    family inet {
        filter MF-classification-and-rl {
            term voice {
                from {
                    source-address {
                        10.1.1.11/32;
                    }
                }
                then {
                    policer voice;
                    forwarding-class expedited-forwarding;
                    accept;
                }
            }
            term server {
                from {
                    source-address {
                        10.1.1.254/32;
                    }
                }
                then {
                    forwarding-class assured-forwarding;
                    accept;
                }
            }
            term any {
                then {
                    forwarding-class best-effort;
                    accept;
```

```
                }
            }
        }
    }
}
class-of-service {
    interfaces {
        ge-3/0/7 {
            unit 0 {
                scheduler-map voice+server+be;
                shaping-rate 10m;
            }
        }
    }
    scheduler-maps {
        voice+server+be {
            forwarding-class assured-forwarding scheduler server;
            forwarding-class expedited-forwarding scheduler voice;
            forwarding-class network-control scheduler nc;
            forwarding-class best-effort scheduler be;
        }
    }
    schedulers {
        voice {
            priority strict-high;
        }
        server {
            transmit-rate 8m;
            priority medium-high;
        }
        be {
            transmit-rate remainder;
            priority low;
        }
        nc {
            transmit-rate percent 5;
            priority high;
        }
    }
}
interfaces {
    ge-3/0/0 {
        unit 0 {
            family inet {
                filter {
                    input MF-classification-and-rl;
                }
            }
        }
    }
    ge-3/0/7 {
       per-unit-scheduler;
    }
}
```

### Out-of-Conformance Traffic

Continuing to build on the previous example, traffic sent to the strict-high queue can not only be rate-limited, but also classified as out of conformance. When extra bandwidth is available, non-conforming traffic can use the bandwidth, but as soon as some congestion is detected, the out-of-conformance traffic will be dropped. The queue bandwidths and configurations are identical to those of the previous example. A stacked policer is added to classify traffic in excess of 1 Mbps and drop traffic over 2 Mbps.

```
firewall {
    policer voice {
        if-exceeding {
            bandwidth-limit 1m;
            burst-size-limit 6k;
        }
        then out-of-profile;
    }
policer voice-excess {
        if-exceeding {
            bandwidth-limit 2m;
            burst-size-limit 6k;
        }
        then discard;
    }

    family inet {
        filter MF-classification-and-rl {
            term voice {
                from {
                    source-address {
                        10.1.1.11/32;
                    }
                }
                then {
                    policer voice;
                    forwarding-class expedited-forwarding;
                    next term;
                }
            }
term voice-excess {
                from {
                    source-address {
                        10.1.1.11/32;
                    }
                }
                then {
                    policer voice-excess;
forwarding-class expedited-forwarding;
                   accept;
                }
            }
            term server {
                from {
                    source-address {
                        10.1.1.254/32;
                    }
                }
                then {
                    forwarding-class assured-forwarding;
                    accept;
```

```
                }
            }
            term any {
                then {
                    forwarding-class best-effort;
                    accept;
                }
            }
        }
    }
}
interfaces {
    ge-3/0/0 {
        unit 0 {
            family inet {
                filter {
                    input MF-classification-and-rl;
                }
            }
        }
    }
}
```

### Changing the Default Forwarding Classes

Previous examples use the preconfigured J-series forwarding classes. Even though these are sufficient in most scenarios, there are some cases, most notably when more than four forwarding classes are needed, in which they need to be changed. This example explains how to add a queue and configure the five resulting queues with the following parameters.

**Table 5. Customized Traffic Classes Example**

| Queue Number | Forwarding Class | Bandwidth | Priority |
|---|---|---|---|
| 0 | best-effort | 500 Kbps | low |
| 1 | voice | - | strict-high |
| 2 | data-high | 5 Mbps | medium-high |
| 3 | network-control | 500 Kbps | high |
| 4 | data-low | 3 Mbps | medium-low |

```
class-of-service {
forwarding-classes {
    queue 1 voice;
    queue 2 data-high;
    queue 4 data-low;
}
interfaces {
    ge-3/0/7 {
        unit 0 {
            scheduler-map voice+server+be;
            shaping-rate 10m;
        }
    }
}
scheduler-maps {
    voice+server+be {
        forwarding-class data-high scheduler server;
        forwarding-class voice scheduler voice;
```

```
        forwarding-class network-control scheduler nc;
        forwarding-class best-effort scheduler be;
        forwarding-class data-low scheduler users;
    }
}
schedulers {
    voice {
        priority strict-high;
    }
    server {
        transmit-rate 5m;
        priority medium-high;
    }
    be {
        transmit-rate remainder;
        priority low;
    }
    nc {
        transmit-rate percent 5;
        priority high;
    }
    users {
        transmit-rate 3m;
        priority medium-low;
    }
}
}
```

User defined forwarding classes override the default classes. The "show class-of-service forwarding-class" command will show the new forwarding classes (or the default classes if no user defined classes exist).

test@J2320-1# run show class-of-service forwarding-class

| Forwarding Class | ID | Queue |
|---|---|---|
| **best-effort** | 0 | 0 |
| **voice** | 1 | 1 |
| **data-high** | 2 | 2 |
| **netword-control** | 3 | 3 |
| **data-low** | 4 | 4 |

After definition, the newly created classes are used just like the default ones. For example, the previous MF classification filter can be redefined using the newly created forwarding classes.

```
firewall {
family inet {
    filter MF-classification-and-rl {
        term voice {
            from {
                source-address {
                    10.1.1.11/32;
                }
            }
            then {
                policer voice;
                forwarding-class voice;
                accept;
            }
```

```
        }
        term server {
            from {
                source-address {
                    10.1.1.254/32;
                }
            }
            then {
                forwarding-class data-high;
                accept;
            }
        }
        term any {
            then {
                forwarding-class best-effort;
                accept;
            }
        }
    }
}
}
```

### Configuring Drop Profiles

In this example, a stacked policer is used to assign traffic a low drop priority when the transmit rate is under 1 Mbps, a high drop priority when the transmit rate is between 1 and 2 Mbps, and an absolute drop for any traffic in excess of 2 Mbps.

Traffic from a specific host will be sent to the assured-forwarding queue which applies two drop profiles, one for low and one for high drop priority traffic.

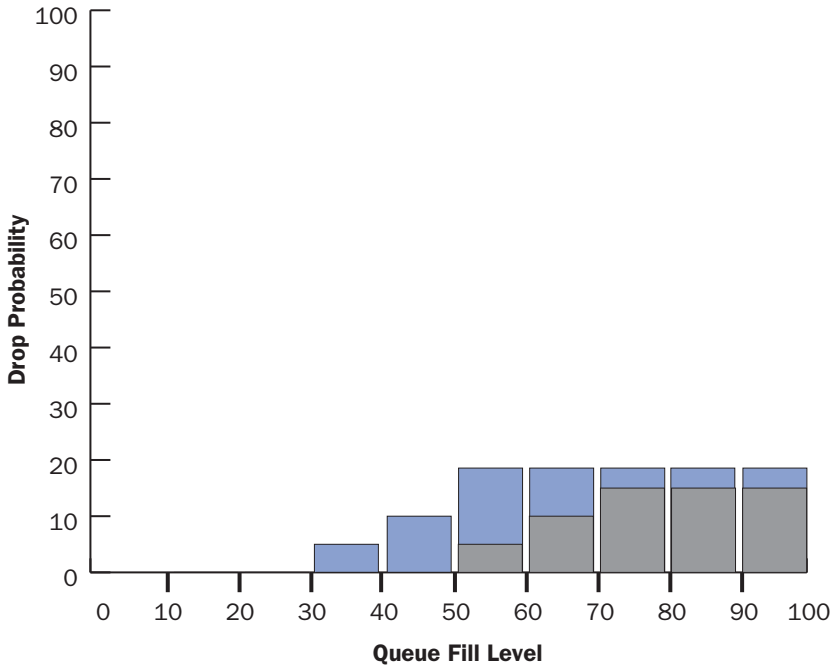


*Figure 9: Drop Profiles for Assured Forwarding Traffic Class*

```
interfaces {
    ge-3/0/0 {
unit 0 {
            family inet {
                filter {
                    input MF-classification-and-rl;
                }
            }
        }
    }
}
class-of-service {
    drop-profiles {
        low {
            fill-level 50 drop-probability 5;
            fill-level 60 drop-probability 10;
            fill-level 70 drop-probability 15;
        }
        high {
            fill-level 30 drop-probability 5;
            fill-level 40 drop-probability 10;
            fill-level 50 drop-probability 20;
        }
    }
    interfaces {
        ge-3/0/7 {
            unit 0 {
                scheduler-map server;
                shaping-rate 10m;
            }
        }
    }
    scheduler-maps {
        server {
            forwarding-class assured-forwarding scheduler data-high;
        }
    }
    schedulers {
       data-high {
            transmit-rate remainder;
            priority low;
            drop-profile-map loss-priority low protocol any drop-profile low;
            drop-profile-map loss-priority high protocol any drop-profile high;
        }
    }
}
firewall {
    policer high-drop-priority {
        if-exceeding {
            bandwidth-limit 1m;
            burst-size-limit 6k;
        }
        then loss-priority high;
    }
    policer over-2M {
        if-exceeding {
            bandwidth-limit 2m;
```

```
            burst-size-limit 6k;
        }
        then discard;
    }
    family inet {
        filter MF-classification-and-rl {
            term host-10.1.1.20 {
                from {
                    source-address {
                        10.1.1.20/32;
                    }
                }
                then {
                    policer high-drop-priority;
                    loss-priority low;
                    forwarding-class assured-forwarding;
                    next term;
                }
            }
            term host-10.1.1.20-discard {
                from {
                    source-address {
                        10.1.1.20/32;
                    }
                }
                then {
                    policer over-2M;
                    accept;
                }
            }
            term any {
                then accept;
            }
        }
    }
}
```

### Rewriting Rules

This final example shows how it is possible to change both the 802.1p priority bits of VLAN tagged traffic as well as the DSCP bits of IP packets. Continuing with the previous example, after traffic from host 10.1.1.20 has been classified, assigned to a forwarding class, and given a drop priority, traffic will be marked according to the following table:

**Table 6. Rewrite Rule Example**

| Forwarding Class | Drop Priority | DSCP | 802.1p |
|---|---|---|---|
| assured-forwarding | low | 001010 (af11) | 111 |
| assured-forwarding | high | 001110 (af13) | 001 |

The only modification to the configuration shown in the previous example is the addition of the rewrite rules and subsequent association with interfaces.

```
class-of-service {
interfaces {
    ge-3/0/7 {
        unit 0 {
            rewrite-rules {
                dscp example-8.12;
                ieee-802.1 example-8.12;
            }
        }
    }
}
rewrite-rules {
    dscp example-8.12 {
        forwarding-class assured-forwarding {
            loss-priority low code-point 001010;
            loss-priority high code-point 001110;
        }
    }
    ieee-802.1 example-8.12 {
        forwarding-class assured-forwarding {
            loss-priority low code-point 111;
            loss-priority high code-point 001;
        }
    }
}
}
```

## Summary

Juniper Networks J-series services routers provide a comprehensive set of QoS features, which provide network administrators the flexibility needed to design and support multi-service networks. These features include the ability to mark, rate-limit, queue, schedule, and selectively drop packets as traffic transverses the device. Together with the ability to monitor end-to-end service, J-series services routers faciliate the deployment of data, voice and video services where performance has to be guaranteed across the network.

## About Juniper Networks

Juniper Networks, Inc. is the leader in high-performance networking. Juniper offers a high-performance network infrastructure that creates a responsive and trusted environment for accelerating the deployment of services and applications over a single network. This fuels high-performance businesses. Additional information can be found at www.juniper.net.

*To purchase Juniper Networks solutions, please contact your Juniper Networks sales representative at 1-866-298-6428 or authorized reseller.*

CORPORATE SALES HEADQUARTERS
Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089 USA
Phone: 888.JUNIPER (888.586.4737)
or 408.745.2000
Fax: 408.745.2100
www.juniper.net

EMEA HEADQUARTERS
Juniper Networks Ireland
Airside Business Park
Swords, County Dublin, Ireland
Phone: 35.31.8903.600
Fax: 35.31.8903.601

APAC HEADQUARTERS
Juniper Networks (Hong Kong)
26/F, Cityplaza One
1111 King's Road
Taikoo Shing, Hong Kong
Phone: 852.2332.3636
Fax: 852.2574.7803