QoS Scheduling

This chapter outlines the process of selecting the next packet to exit an interface and when it should happen (henceforth termed Scheduling). The topic of scheduling exploits the following commands: **priority**, **bandwidth**, **bandwidth remaining**, **shape** and **fair-queue**. Using these commands we can apportion bandwidth when congestion exists and ensure that applications receive the treatment they need to operate over the network.

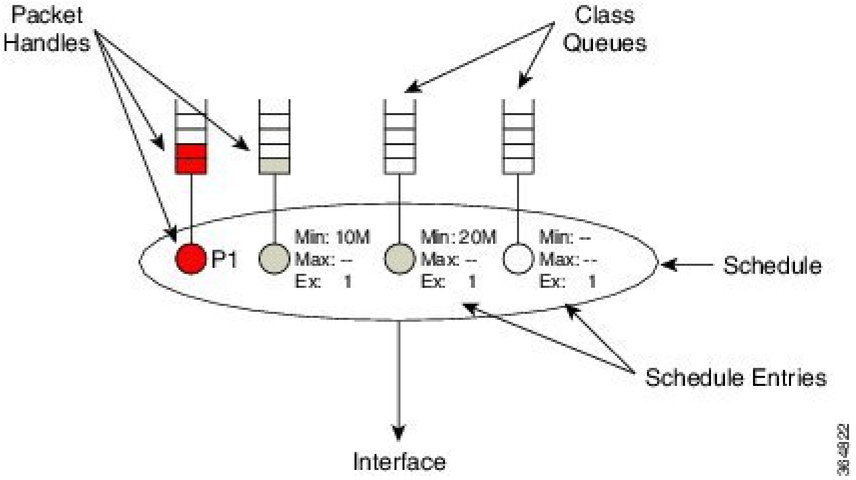
Specifically, this chapter will focus on flat policies attached to physical interfaces. The information presented here should ground your understanding of hierarchical scheduling concepts discussed in int following chapters.

About QoS Scheduling

Definitions

In this section we define core "scheduling" terms.

Figure 1: Scheduling Definitions



Packet Handle

When a router is prepared to forward a packet, it places a *packet handle*, representing that packet, in one of the egress queues. This handle holds information like the length of the packet and the location of the packet in memory.

Class Queues

When egress QoS is configured, a *class queue* is created for each class where we configure a queuing action. Similarly, we create an *implicit class-default queue* for any traffic not matching one of the explicitly-created queuing classes. If you configure a class with only non-queuing actions (e.g., a class with only marking configured), "matching" packets will be enqueued in the class-default queue.

Schedule

You should view a *schedule* (scheduler) as the decision maker. By selecting the packet handle, the schedule chooses which packet should next exit and when to send it. In the diagram above the "oval" represents a single schedule that selects a packet from one of the class queues.

Note An individual schedule is created for each interface.

Schedule Entry

For a schedule to choose between queues it needs to know each queues' expected treatment. We store this type of information in a *schedule entry*. For example, by configuring a queuing command (e.g. **bandwidth 10 Mbps**) you are setting the schedule entry.

The schedule entry also stores the internal state like the last time a packet was transmitted from that queue and the current packet handle, if any, from that queue.

Two types of schedule entries include the following: Priority Queues, on page 15, and Bandwidth Queues, on page 22.

How Schedule Entries are Programmed

In this section we provide a brief introduction to the parameters that are configured within a schedule entry. The actual commands will be covered in greater detail later in this chapter.

Firstly, a schedule entry is configured as either a *priority entry* or *bandwidth entry* (*priority queue* or *bandwidth queue*).

In the descriptions that follow you will see that priority entries can be further divided into *P1 entries* or *P2 entries*. You configure a P1 entry (the default) with either the **priority** or **priority level 1** command. Similarly, you use the **priority level 2** command to configure a P2 entry.

A bandwidth entry has three distinct parameters: *minimum rate* (Min), *maximum rate* (Max) and *excess weight* (drawn as "Ex" in illustrations).

Note The scheduler for a ASR 1000 Series Aggregation Services Router is often described as a three-parameter scheduler.

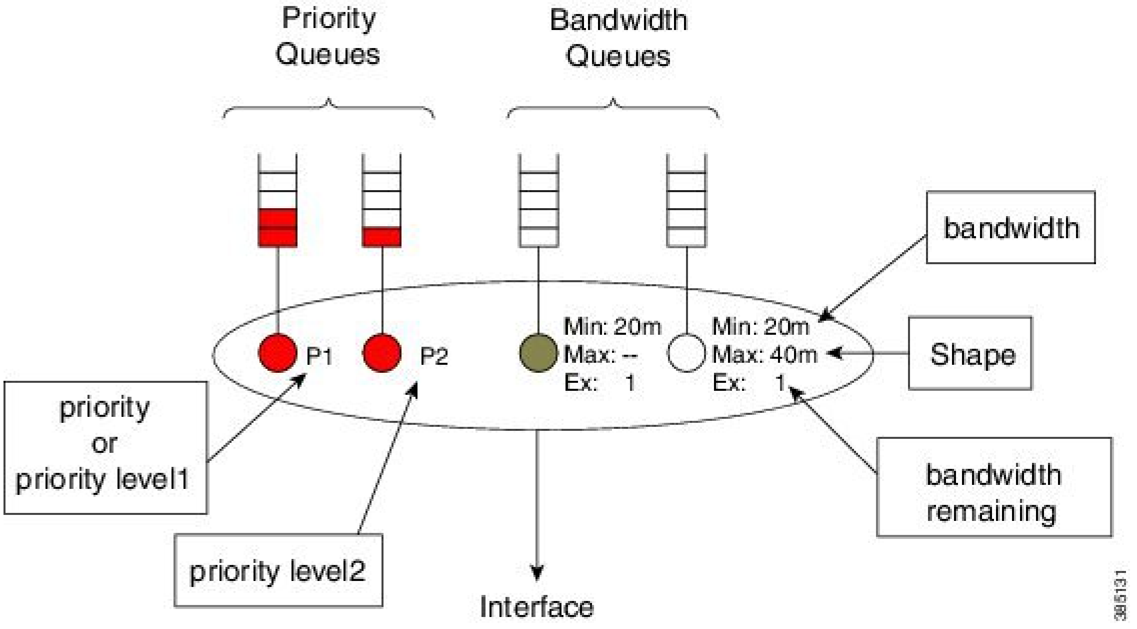
The **Min (minimum rate)** entry allocates a minimum bvandwidth guaranteed amount of throughput to a queue. The Min entry is configured with the **bandwidth** command and is not set unless explicitly configured. IOS configuration checking attempts to ensure that a schedule will always have sufficient bandwidth to honor any configured Min rates. Servicing queues based on monitoring throughput vs. a preconfigured target rate is sometimes referred to as *real time scheduling* (refer to Scheduler's Representation of Time, on page 33).

The **Max (maximum rate)** entry establishes a ceiling on the amount of throughput a queue can receive. The Max entry is configured using the **shape** command and is not set unless explicitly configured. Understand that Max sets a ceiling on the throughput of a queue but does not in itself guarantee any throughput to that queue.

The **Ex (excess weight)** entry mandates how queues will compete for any bandwidth available after Priority and Min guarantees have been met (*excess bandwidth*, or available bandwidth that is not guaranteed to, or not used by, priority and bandwidth guarantees). We configure Excess Weight with the **bandwidth remaining** command and unless explicitly configured, it defaults to 1. Excess *bandwidth sharing* is proportional to a queue's Excess Weight (sometimes referred to as *virtual time scheduling*, because no rates are configured and relative behavior alone is significant). For reflections on bandwidth sharing, see How Schedule Entries are Programmed, on page 3.

The following diagram summarizes what is presented above (the commands to set each schedule entry).

Figure 2: IOS Commands to Set Schedule Entries



Schedule Operation

How a schedule determines the packet sequence may be summarized as follows:

Note After each packet is forwarded, we return to step 1.

1. If the P1 queue is not empty, send the P1 packets.
2. If the PI queue is empty but the P2 queue is not, send the P2 packets.
3. Provided all priority queues are empty, the schedule services any queues with a minimum bandwidth guarantee (Min) and continues to service such queues until the guarantees are met. To ensure fairness, the scheduler will pick between queues with minimum guarantees by selecting the eligible queue, a queue that has not exceeded the bandwidth guarantee and has been waiting longest.
4. What if priority queues are empty and all bandwidth guarantees have been satisfied? Any excess bandwidth is distributed between queues that still require service until either all bandwidth is exhausted or a given queue has reached a maximum configured bandwidth. The Ex configured in that queue's schedule entry, dictates the share each queue will receive of this excess bandwidth.

Schedule Operation: Without a Shaper

The following example illustrates how a schedule operates and how it determines the bandwidth each queue will receive for a given offered load.

Before diving into the example we need to introduce the concept of *priority queue admission control*. In the previous description of schedule operation you will notice an absence of rates regarding how the schedule deals with priority queues; the schedule simply selects the priority queue whenever it contains a packet.

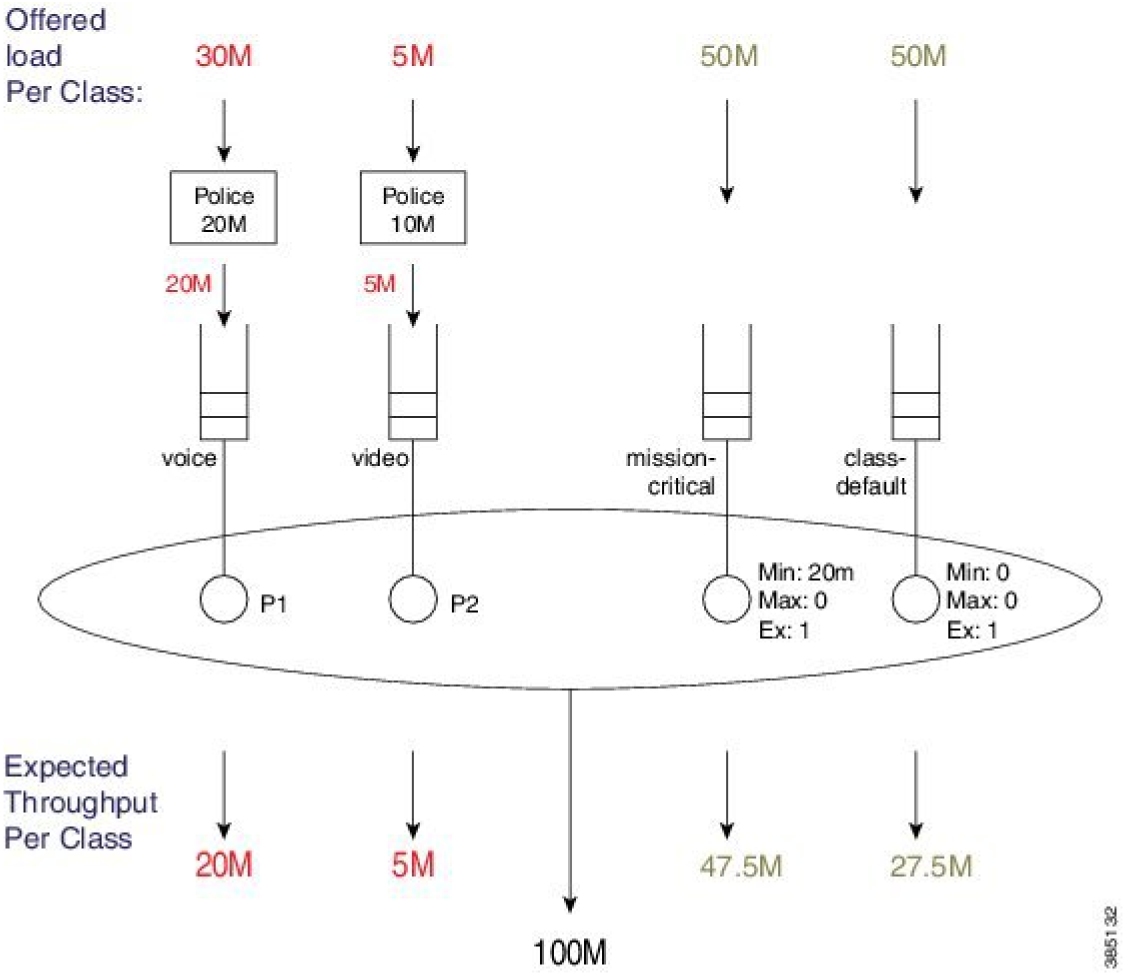
To prevent a priority queue (class) from starving other queues of service, we can use a policer to limit the consumable bandwidth. Such a policer restricts the rate at which packets can be enqueued in that queue.

In the following example, we attach a policy to a 100 Mbps interface:

```
policy-map scheduling-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
  class class-default
```

Note Bandwidth is configured in Kbps. While **police** and **shape** commands support a postfix to specify the unit, the **bandwidth** command does not.

Figure 3: Scheduling Operation



The loads offered to each class are shown at the top of the figure: 30M, 5M, 50M, and 50M. We have applied policers (20M and 10M) to the priority queues.

30 Mbps is offered to the voice class, which first traverses a 20 Mbps policer, enqueuing 20 Mbps to the P1 queue. Because we always service this queue first, all 20 Mbps enqueued will be forwarded.

5 Mbps is offered to the video class (which all transits the 10 Mbps policer) and 5 Mbps is enqueued to the video queue. As 80 Mbps (100 Mbps - 20 Mbps) bandwidth is still available, all 5 Mbps will be forwarded.

After servicing priority queues, we advance to any queues with an explicit Min bandwidth guarantee. The mission-critical class has a Min of 20 Mbps so it will receive at least that amount of throughput.

The available excess bandwidth is 55Mbps (100 Mbps - 20 Mbps - 5 Mbps - 20 Mbps). Both the class-default and mission-critical classes have default excess weights of 1, so each receives an equal share of the available bandwidth, (55Mbps/2 =) 27.5 Mbps.

The mission-critical class will observe a total throughput of 47.5 Mbps (20 Mbps + 27.5 Mbps).

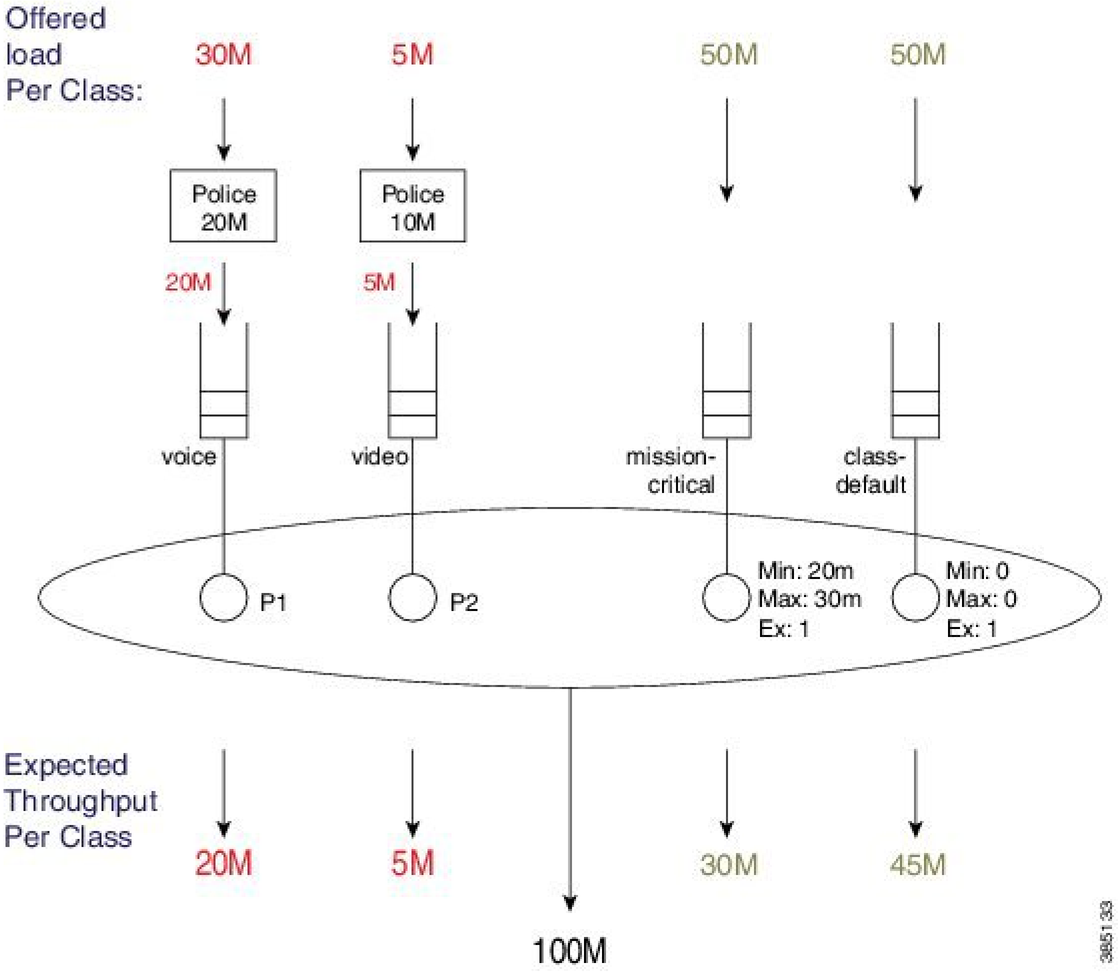
Schedule Operation: With a Shaper

Let's modify the configuration slightly - we will add a Max value (configure a shaper) to the mission-critical class:

```
policy-map scheduling-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
    shape average 30m
```

Note We excluded class-default in the policy definition - it is always there whether or not we explicitly define it.

Figure 4: Scheduling Operation



The loads offered to each class is exactly as before: 30M, 5M, 50M and 50M.

30 Mbps is offered to the voice class, which first passes through a 20 Mbps policer, enqueing 20 Mbps to the P1 queue. We always service this queue first, so all 20 Mbps enqueued will be forwarded.

5 Mbps is offered to the video class (which all passes through the 10 Mbps policer) and 5 Mbps is enqueued to the P2 queue. As 80 Mbps (100 Mbps - 20 Mbps) bandwidth is still available, all 5 Mbps will be forwarded.

After servicing priority queues, we advance to any queues with an explicit Min. The mission-critical class has a bandwidth guarantee of 20 Mbps so it will receive at least that amount of throughput.

The available excess bandwidth is 55 Mbps (100 - 20 - 5 - 20 Mbps). Both the class-default and mission-critical classes have default Ex's 1, so each receives an equal share of the available bandwidth. From the Excess bandwidth sharing "rule," where in bandwidth is proportional to a queue's Ex, each class receives a 27.5 Mbps share. (For more information on this "rule," refer to How Schedule Entries are Programmed, on page 3.)

Based on the bandwidth guarantee and bandwidth sharing, the mission-critical queue would receive 47.5 Mbps (20 + 27.5 Mbps). However, the queue cannot use this much bandwidth because the Max configured shape rate is set to 30 Mbps (recall that Max is set to 0 in the previous example). Consequently, the queue uses 30 Mbps (out of the 47.5 Mbps received from bandwidth sharing) and the additional 17.5 Mbps of bandwidth returns to the excess pool.

As class-default is the only queue still requesting bandwidth, it has no competition and can consume this extra 17.5 Mbps, increasing its total throughput to 45 Mbps.

Note This example demonstrates how bandwidth is never wasted - scheduling will continue to sort through eligible queues and apportion bandwidth until one of the following applies:

- Each queue is empty.
- All Max values have been reached.
- All bandwidth has been consumed.

Configuring Rates and Burst Parameters

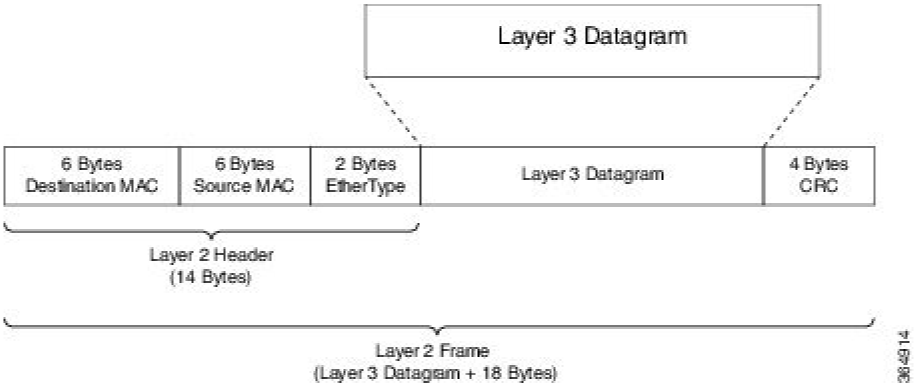
What's Included in Scheduling Rate Calculations (Overhead Accounting)

In the discussion of Schedule Operation, on page 4, you will notice that Min and Max are configured in bits per second. But what do these rates include? The short answer is that a schedule includes the Layer 3 datagram and Layer 2 header lengths but neither CRC nor inter-packet overhead.

Layer 3 Datagram

To clarify, let's imagine transporting an IP datagram over a GigabitEthernet link.

Note Henceforward, we will refer to a "schedule's perception of the packet length" as the *scheduling length*.

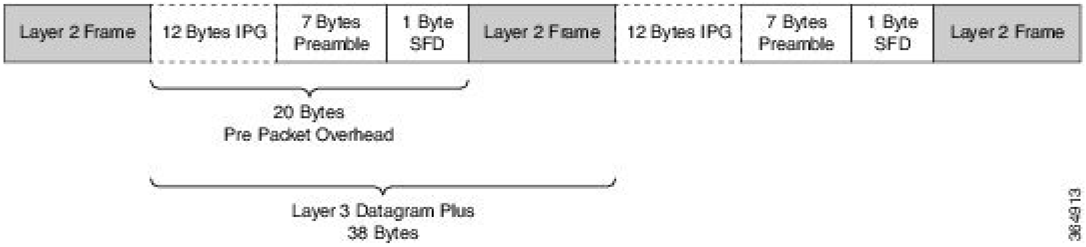


Ethernet Overhead

To transport the datagram on a GigabitEthernet link we first need to encapsulate it correctly in an Ethernet frame. This process adds 14 bytes of Layer 2 header and an additional 4 bytes of CRC (i.e., total of 18 bytes for encapsulation).

Consider what happens when this Layer 2 frame is transmitted over the physical medium. Ethernet requires a minimum *inter packet gap* (IPG) equal to the transmit time for 12 bytes of data, 7 bytes of preamble, and a *single-byte start-of-frame delimiter* (SFD), for a total pre-packet overhead of 20 bytes:

Figure 5: Ethernet Overhead



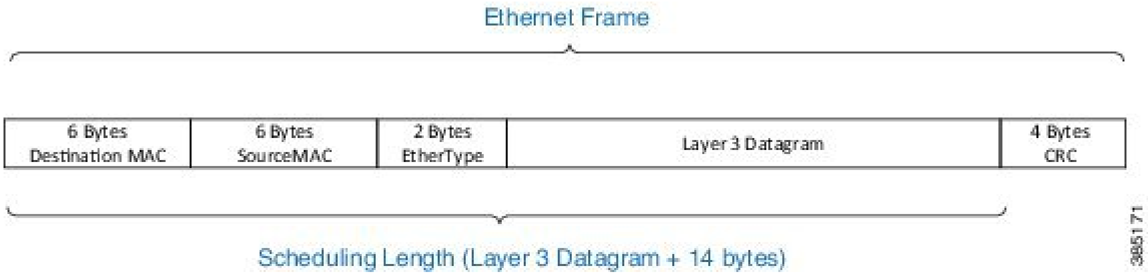
So, if you send multiple Ethernet frames sequentially, the total per-packet overhead for each Layer 3 datagram is an additional 38 bytes (encapsulation (18 bytes) + Ethernet inter-packet overhead (20 bytes)). For example, if you were to send 100 byte IP datagrams at line rate on a GigabitEthernet link, the expected throughput in packets per second would be:

```
  Linerate / Bits Per Byte / ( Layer 3 length + Per Packet Overhead) = Packets Per Second
  1 Gbps / 8 / ( 100 + 38 ) = 905, 797 pps
```

Scheduling Length

From the scheduler's viewpoint, the packet's length is Layer 3 datagram + Layer 2 header (14 bytes on a GigabitEthernet interface):

Figure 6: Scheduling Length



Now consider a 500-Mbps shaper configured on a GigabitEthernet interface. (Recall that shaping is the process of imposing a maximum rate of traffic while regulating the traffic rate in such a way that downstream devices are not subjected to congestion.) As in the previous example, we will send all 100-byte IP datagrams to the scheduler, resulting in a "scheduling length" of 114 bytes (100 byte (datagram) + 14 byte (Ethernet Layer 2 header)). According to the following formula, the anticipated throughput would now be:

```
  Shaper Rate / Bits per Byte / (Layer 3 length + Layer 2 header length)
= Packets Per Second
  500 Mbps / 8 / (100 + 14) = 548,246 pps
```

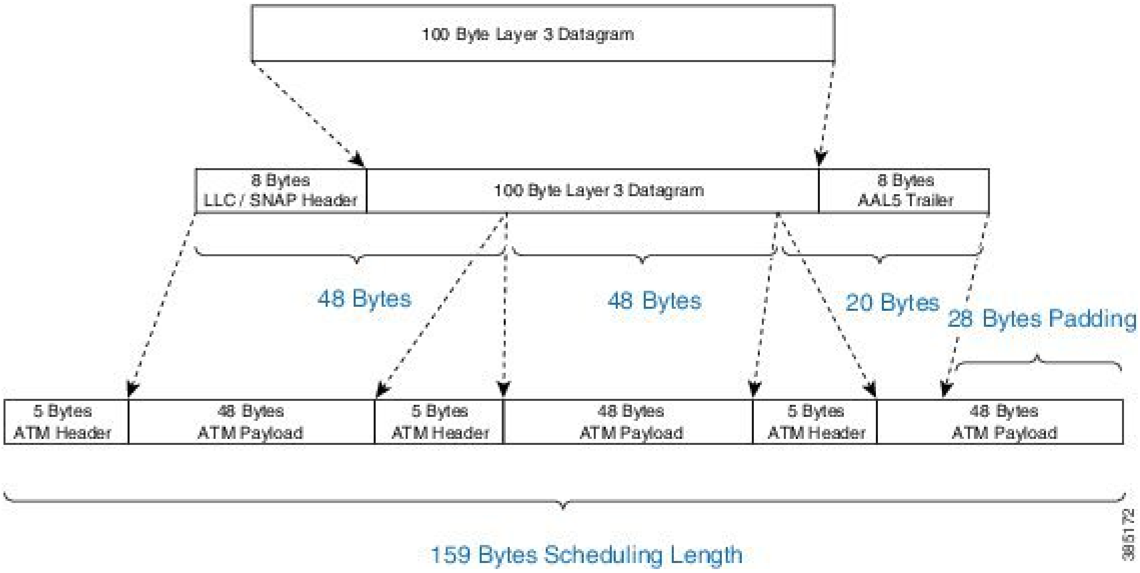
Observe that 100% of linerate (all 100 byte datagrams) was 905,797 packets per second but shaping to 500 Mbps (all 100 byte datagrams) yielded a throughput of 548,246 packets per second. Obviously, this is considerably more than 50% of physical capacity. When specifying rates to apportion bandwidth, be aware that rates do not include all overhead required to transport that packet.

Scheduler on an ATM Interface

When a queuing policy (scheduling policy) is attached to an ATM VC, scheduling rates in that policy are inclusive of all cell tax. This differs from a policer configured in such a policy that only includes the AAL5 header.

For example, consider a 100-byte datagram sent over an ATM VC configured with AAL5 SNAP encapsulation. A router will add an 8-byte LLC/SNAP header to the datagram, yielding a *policing length* of 108 bytes (analogous to a scheduling length of 114 bytes for a 100 byte IP datagram. (See What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 9.) (For further details on policing length, refer to Priority Policing Length, on page 21.)

Figure 7: Scheduler on ATM Interface



To convey the packet, the router must also add an 8-byte AAL5 trailer (to the policing length) and then spilt the packet into ATM cells. To transport this packet we require 3 ATM cells, each carrying 48 bytes of the packet. We pad the third cell such that it also has a 48-byte payload.

Each of these 3 cells is 53 bytes in length (48 bytes packet + 5 byte ATM Header), which means that the scheduling length of the 100-byte datagram would be 159 bytes (3 cells x 53 bytes per cell).

Scheduler on a Logical Interface

In the Policing chapter we discuss how policer overhead accounting may differ depending on whether the policy is attached to a physical or logical interface (see Policer on Logical Interface). This situation does not apply to a scheduler. Although a policy is attached to a logical interface (a tunnel interface) we must complete all processing and add any necessary headers before we enqueue the packet to egress a physical interface. Because we know the final length of the packet at the time of enqueue, we can set the scheduling length accordingly at that time.

Scheduler Overhead Accounting Adjustment

In prior sections, we described what is included by default in scheduler rate calculations. Occasionally, however, a user might want behavior to differ from the default.

For example, we hear that users want to express rates as physical bandwidth that would be consumed on the link. For an Ethernet interface, you would need to include the 4-byte CRC and 20-bytes inter-packet overhead required by each packet.

We also hear from service providers who want to charge their customers for traffic throughput at Layer 3 rates. The datagram's length remains constant as a packet traverses different interface types or encapsulating protocols, making it easier for users to understand. In this instance, we would not include the Layer 2 header length in shape rate calculations.

Note Changing overhead accounting may impact the network elsewhere. For example, if we use a policer for network admission control, we typically configure a shaper on customer premises equipment connecting to that network. The shaper and policer should have the same view of what is included in CIR.

Scheduler Account Option

The scheduler account option (the **account** keyword) allows you to specify a number of bytes that should be added or removed from the default "scheduling length" per packet to achieve the desired behavior. You can add or subtract at most 63 bytes per packet. This option is supported on the **shape** and **bandwidth** commands.

In the following example, we apply a shaper on an Ethernet interface and we want to include all overhead such that the shaper will cap throughput at 50% of the actual physical bandwidth. By adding 24 bytes per packet we "cover" the 4 byte CRC and 20-bytes inter-packet overhead:

```
policy-map ethernet-physical-example
  class class-default
    shape average percent 50 account user-defined 24
```

Note With overhead accounting, we must account for *hierarchical policies*. If a parent shaper is configured with the account option, any child shapers or bandwidth guarantees will also inherit the same adjustment as specified in the parent policy.

In the chapter on hierarchical scheduling, we will observe how to use shapers to condition traffic for remote links and to use child polices for apportioning bandwidth within that shape rate. In that use case, the encapsulation on the remote link may differ from the encapsulation on the sending device (e.g. an enterprise hub router connected to the network with an Ethernet interface sends traffic to a branch connected with a T1 interface). If the T1 link were using HDLC encapsulation each datagram would have 4 bytes of Layer 2 headers on that link. On the Ethernet, however, each packet would have 14 bytes of Layer 2 headers. The account option can be used to shape and schedule packets as they would appear on that remote link. That is, remove 14 bytes from the scheduling length as Ethernet headers are no longer present and then add 4 bytes to the scheduling length to represent the HDLC Layer 2 overhead.

Overhead Accounting Adjustment (Predefined Options)

In addition to specifying a number of bytes to add or subtract (see the following table), the CLI also offers some predefined options with which you can specify remote encapsulation. The current predefined options are based on broadband use cases, assuming that we send (or receive) traffic on an Ethernet interface to a DSLAM elsewhere in the network. Although we are encapsulating in Ethernet frames that include Dot1Q or Q-in-Q, the DSLAM receives some form of ATM encapsulation. We want the shaper to condition traffic to mirror how it would appear after DSLAM. In each case we would also add cell-tax to the scheduling length.

Table 1: Table of Predefined Options for Overhead Accounting Adjustment

CLI	Value (dot1q/qinq)	ATM	Details (dot1q/qinq)
account dot1q|qinq aal5 mux-1483routed	-15/-19	yes	dot1q: 3 byte 1483 routed - 18 byte dot1q qinq: 3 byte 1483 routed - 22 byte qinq
account dot1q|qinq aal5 mux-dot1q-rbe	0/-4	yes	dot1q: 0 byte mux_rbe + 18 byte dot1q - 18 byte dot1q qinq: 0 byte mux_rbe + 18 byte dot1q - 22 byte qinq
account dot1q|qinq aal5 mux-pppoa	-22/-26	yes	dot1q: 2 byte mux_pppoa - 6 byte pppoe - 18 byte dot1q qinq: 2 byte mux_pppoa - 6 byte pppoe - 22 byte dot1q
account dot1q|qinq aal5 mux-rbe	-4/-8	yes	dot1q: 0 byte mux_rbe + 14 byte 802.3 - 18 byte dot1q qinq: 0 byte mux_rbe + 14 byte 802.3 - 22 byte qinq
account dot1q|qinq aal5 snap-1483routed	-12/-16	yes	dot1q: 6 byte snap 1483 routed - 18 byte dot1q qinq: 6 byte snap 1483 routed - 22 byte qinq
account dot1q|qinq aal5 snap-dot1q-rbe	10/6	yes	dot1q: 10 byte snap_rbe + 18 byte dot1q - 18 byte dot1q qinq: 10 byte snap_rbe + 18 byte dot1q - 22 byte qinq
account dot1q|qinq aal5 snap-pppoa	-20/-24	yes	dot1q: 4 byte snap_pppoa - 6 byte pppoe - 18 byte dot1q qinq: 4 byte snap_pppoa - 6 byte pppoe - 22 byte qinq

CLI	Value (dot1q/qinq)	ATM	Details (dot1q/qinq)
account dot1q|qinq aal5 snap-rbe	6/2	yes	dot1q: 10 byte snap_rbe + 14 byte 802.3 - 18 byte dot1q qinq: 10 byte snap_rbe + 14 byte 802.3 - 22 byte qinq
account user-defined <value>	<value>	no	
account user-defined <value> atm	<value>	yes	

Imagine that we forwarding Dot1Q encapsulated packets on an Ethernet interface. Imagine further than a downstream DSLAM will:

- receive the packets
- strip the Ethernet and Dot1q headers
- perform AAL5-Mux 1483 routed encapsulation.

Referring to the previous table, DSLAM will remove 18 bytes of Ethernet/Dot1q and add a 3-byte LLC header, generating a -3 bytes change in the scheduling length. (For a schematic, refer to What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 9.)

As the DSLAM is sending over an ATM network, it would add an 8-byte AAL trailer and then split the resulting PDU into 53-byte cells. The ATM value "yes" (with reference to the table) indicates that the router will calculate this cell tax and include that extra overhead in the scheduling length.

```
policy-map atm-example
  class class-default
    shape average 50m account dot1q aal5 mux-1483routed
```

Example - Predefined Overhead Accounting

Imagine we are forwarding Dot1Q encapsulated packets on an Ethernet interface. Imagine further than a downstream DSLAM will:

- receive the packets
- strip the Ethernet and Dot1q headers
- perform AAL5-Mux 1483 routed encapsulation.

Referring to the previous table, DSLAM will remove 18 bytes of Ethernet/Dot1q and add a 3-byte LLC header, generating a -3 bytes change in the scheduling length. (For a schematic, refer to What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 9.)

As the DSLAM is sending over an ATM network, it would add an 8-byte AAL trailer and then split the resulting PDU into 53-byte cells. The ATM value "yes" (with reference to the table) indicates that the router will calculate this cell tax and include that extra overhead in the scheduling length.

```
policy-map atm-example
  class class-default
    shape average 50m account dot1q aal5 mux-1483routed
```

Priority Queues

Priority queuesrepresents a type of schedule entries that enable you to avoid any unnecessary delay in forwarding packets. Through the priority semantic, we can guarantee low latency treatment for applications that are latency and (or) jitter sensitive. As an example, consider Voice over IP (VOIP). Typical VOIP phones have a 30 mS *de-jitter buffer*, allowing them to tolerate a maximum of 30 mS jitter end-to-end across the network.

When you configure a priority queue, you can select one of three ways to control the bandwidth that might be consumed by that traffic class: unconstrained priority queue, conditional policer, or (un-conditional) always-on policer.

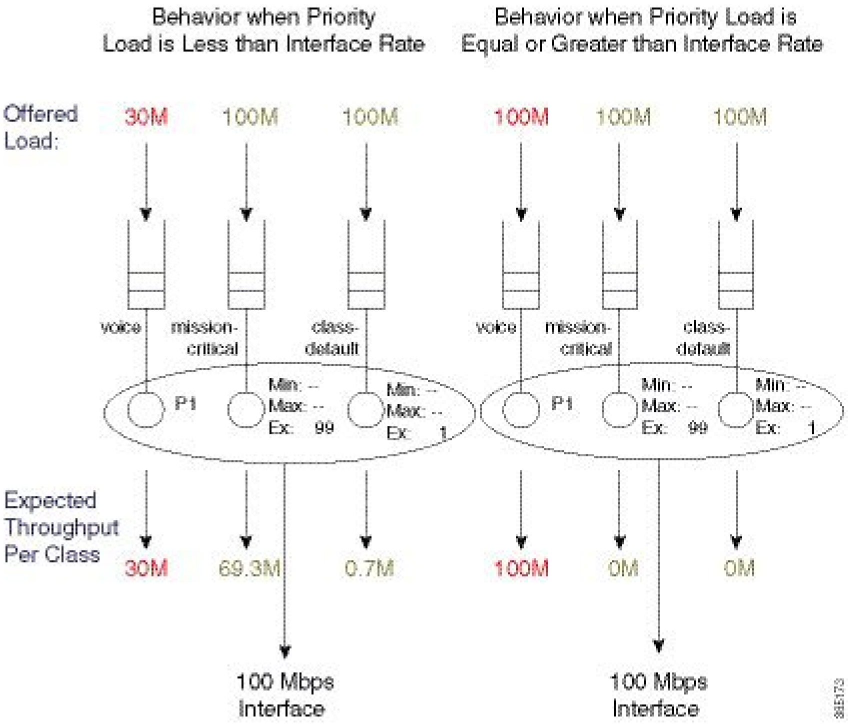
Unconstrained Priority Queue

One way to control bandwidth consumption is with an *unconstrained priority queue* (absolute priority queue), which is a priority queue configured without any limit on the amount of bandwidth that may be consumed by the priority class. To illustrate, consider this example configuration as well as the configured schedule entries in the following figure. Note the lack of a policer for admission control to the priority queue.

```
policy-map absolute_pq_example
  class voice
    priority
  class mission-critical
    bandwidth remaining percent 99
  class class-default
```

```
    bandwidth remaining percent 1
```

Figure 8: Unconstrained Priority Queue



In the example on the left, the actual interface bandwidth capacity (100 Mbps) exceeds the load to the priority queue of the voice class (30 Mbps). This leaves 70 Mbps of excess bandwidth to apportion based on the excess weight (Ex) ratio (99:1; set by the **bandwidth remaining** command).

In the example on the right, the priority load has been increased to 100 Mbps. Because this leaves no excess bandwidth, other queues are starved of service (an expected throughput of 0M).

The take-home is that without admission control on a priority class, a class might consume the entire interface bandwidth and so starve all other service queues. This can cause mission-critical applications to suffer. Moreover, if control messages are in the starved service queue, network instability might result.

So, use unconstrained priority queues with caution. To ensure the priority queue is unable to starve others of service, you might want to consider using alternative bandwidth control systems like Call Admission Control (CAC).

Note You cannot use minimum bandwidth guarantees (as set by the **bandwidth** command) in conjunction with unconstrained priority queues. If the priority queue is capable of consuming all available bandwidth it follows that you can't guarantee any of that bandwidth to other classes. IOS will reject any such configurations.

Priority Queue with Conditional Policer

Another way to control bandwidth consumption is to enter a value with the **priority** command. (See the command page for priority.) This represents a way to handle queue admission control with a conditional policer.

Conditional priority rate limits traffic with a policer only if congestion exists at the parent (policy-map or physical interface) level. This state exists provided more than the configured maximum rate of traffic attempts to move through the class (and/or interface).

The key element is that a conditional policer will only drop packets if the schedule is congested. That is, it will only drop packets when the offered load exceeds the available bandwidth (interface bandwidth in the context of a flat policy-map attached to a physical interface).

A conditional priority class can use more than its configured rate, but only if contention with other classes in the same policy is absent.

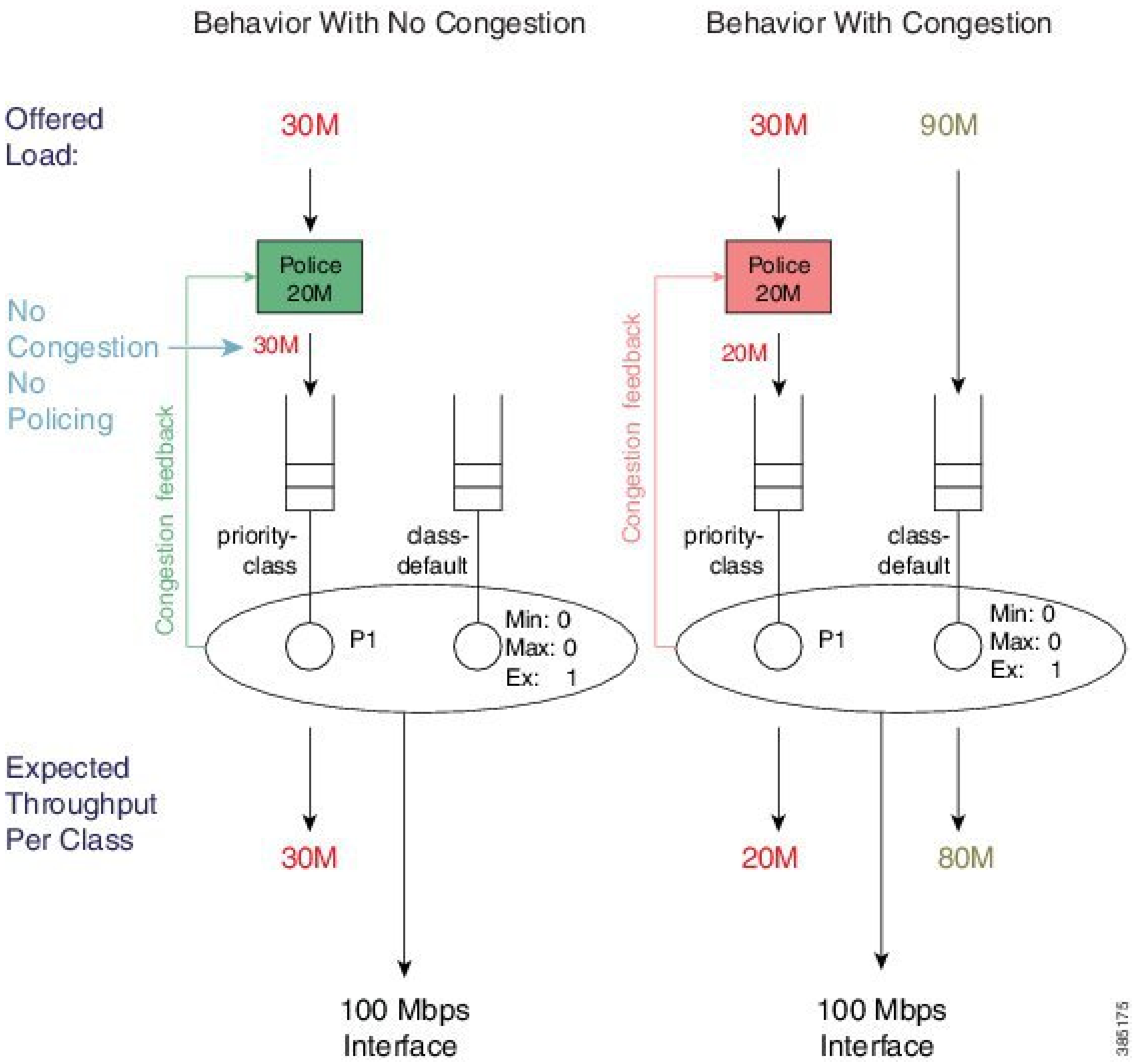
The schedule provides congestion feedback to the policer, which will count every packet in the rate it observes but will suppress the drop action unless congestion exists. So, if no congestion exists, the priority queue may consume whatever bandwidth is available but when congestion occurs the queue will be policed to the configured rate.

```
policy-map conditional_policer_example
  class priority-class
    priority 20000
```

The priority value is configured in kps and although configured with the **priority** command, it does not alter a schedule entry. (Recall that a schedule entry is where we store a queue's expected treatment.) Instead, it configures a policer that will be executed before packets may be enqueued.

The following diagram shows how a conditional policer would function with and without congestion. In this example, we have attached the previous configuration to a 100 Mbps interface.

Figure 9: Priority Queue with Conditional Policer



In the schedule depicted on the left no congestion exists. As the congestion feedback reports no congestion and the policer will enqueue the entire 30 Mbps offered to that class.

In the schedule depicted on the right where congestion exists, the policer will enforce the 20 Mbps rate configured with the **priority** command.

Be aware that a conditional policer has advantages and disadvantages:

Advantage	A priority class may use all bandwidth not currently used by other classes.

Disadvantage	You cannot carefully plan for priority capacity throughout your network if you don’t know the forwarding-capacity of a particular interface. A *true priority service* should have low latency (no queue build-up) and no drops, end-to-end. You experience inconsistent behavior depending on whether or not an interface is congested. If you under-provision the police rate, you may observe intermittent problems with applications using that class and diagnosing the issue might be very difficult.

Note You cannot use conditional policers and policer overhead accounting adjustment concurrently.

Priority Queue with Always on (Unconditional) Policer

The third way to control bandwidth consumption is to use an explicit always-on (i.e., unconditional) policer for queue admission control.

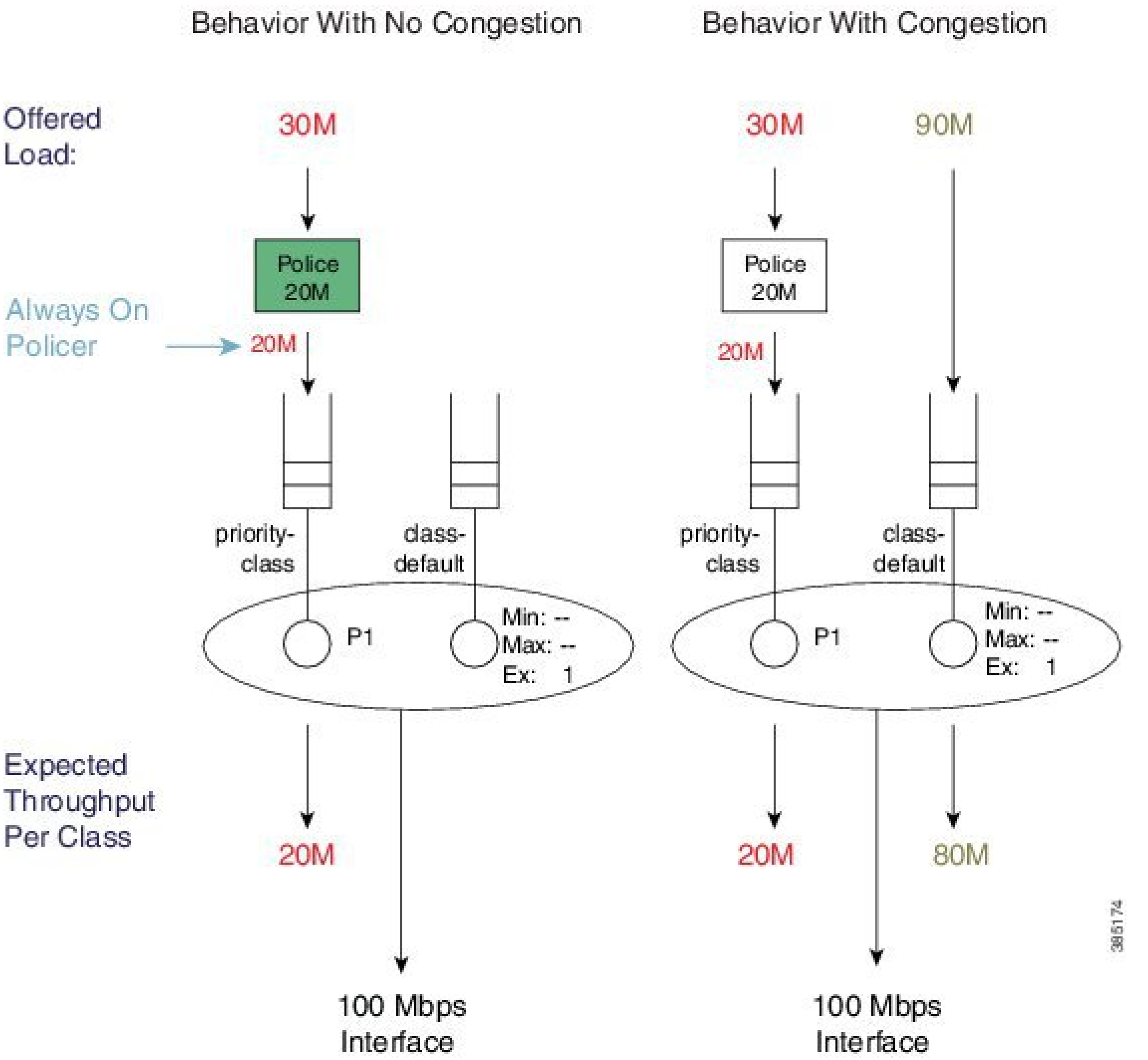
When you configure a priority class with an explicit policing rate, traffic is limited to the policer rate regardless of congestion conditions. That is, even if bandwith is available, the priority traffic cannot exceed the explicit rate.

The following example shows how such a configuration might look:

```
policy-map always_on_policer
  class priority-class
    priority
    police cir 20m
```

The diagram below shows the behavior of such a policer.

Figure 10: Always on Policer



When you configure a priority class with an explicit policing rate, this rate is always enforced. That is, even with sufficient bandwith, priority traffic cannot exceed the explicit rate. This means that you have *deterministic behavior* in your priority service. If a user complains of poor application performance you can look for policer drops in your network and determine if insufficient bandwidth is allocated to the priority service. Applications should have the same experience regardless of whether or not congestion exists.

Priority Queue Burst Considerations

In previous sections we describe how to perform queue admission control for the priority queue using a conditional or always-on policer. Specifically we employ a *single-rate two-color policer*. From the policing chapter we know that policers are implemented using a token bucket scheme that allows for some burst (Single-Rate, Two-Color Policer). Controlling this (burst) allowance is crucial when you use policers in this way.

Bursts that are allowed by the policer may result in a build-up of the priority queue, which will generate latency for a packet that is added to the end of that queue. The packet must wait for the transmission of all preceding packets in the queue before it too can be transmitted. The amount of latency depends on the *serialization delay*, the time taken to transmit those packets on the physical medium.

As multiple packets from a given flow arrive, they may experience different conditions in the priority queue. One might arrive and be enqueued behind a number of packets already waiting and thus experience some latency. The next packet in that same flow may arrive to an empty priority queue and be scheduled immediately. What this means is that any potential latency from priority queue congestion is also potential *jitter* (jitter, potential the variation in the latency of received packets).

The *default burst allowance* for policers in IOS is set at 250 mS for always on policers and 200 mS for conditional policers. If a policer can allow us to enqueue a burst, it follows that these numbers can be almost directly translated into potential jitter. In the introduction to the priority semantic (see Priority Queues, on page 15), we indicated that voice applications can typically tolerate about 30 mS of jitter and 150 mS latency end-to-end across a network. Given the former, we usually try to apportion some of this budget to each node in the network. A simple guideline is to allow a burst tolerance (and thereby potential jitter) of 5-10 mS on any single node.

For example, envisage a priority queue configured with a queue-admission policer at a rate of 2 Mbps and a burst allowance of 5 mS. Calculate the number of bytes we can transmit in 5 mS:

Burst Target
= Police Rate / 8 Bites per Byte * 5 mS
= 2 Mbps / 8 * .005 = 1250 bytes

For an always on policer, the configuration for this example would look like:

```
policy-map always_on_policer_burst_example
  class voice
  priority
  police cir 2000000 1250
```

For a conditional policer, the configuration example would look like:

```
policy-map conditional_policer_burst_example
  class voice
  priority 20000 1250
```

Priority Policing Length

In the section What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 9 we introduced the concept of scheduling length, which is how a scheduler "views" packet length when it is evaluating conformance to a rate. In the Policing chapter we also introduced the similar concept of policing length (What's Included in the Policer-Rate Calculation (Overhead Accounting)). As the rate configured on a priority queue is a policing rate, we will use the policing length when determining conformance to that rate. When a policy is attached to a physical interface, as described in this chapter, the policing and scheduling lengths are identical. To alter the policing length, you can use the policer overhead accounting feature.

Note The **account** keyword is supported with always-on policers but not conditional policers.

Multi-Level Priority Queuing

The Multi-Level Priority Queues (MPQ) feature allows you to configure multiple priority queues for multiple traffic classes by specifying a different priority level for each of the traffic classes in a single service policy-map.

In Schedule Operation, on page 4, we introduced that a priority queue may be P1 or P2. The original intent of this feature was to support voice and video in separate priority queues as each has differing traffic characteristics and jitter tolerance. In particular, voice has a smaller packet size (typically around 80 bytes for voice compared to 1400 bytes for video) and tighter jitter requirements (typically 30 mS whereas non-interactive video may be 100's of mS). So, we would use a P1 queue for voice and a P2 queue for video traffic.

Today many video applications use advanced adaptive codecs and separate the voice and video content into separate streams. Some argue that video traffic is now TCP-like in its behavior and does better in bandwidth queues. Interactive video on slower links may still require a P2 queue.

To configure multilevel priority queuing you must use the **level** keyword in the **priority** command. This feature is supported with conditional policers, always-on policers and absolute priority queues.

Here is an example of a multilevel priority queue with conditional policers:

```
policy-map multilevel-example2
  class voice
    priority level 1 5000 3125
  class video
    priority level 2 10000 12500
```

Here is an example of a multilevel priority queue configuration with always-on policers:

```
policy-map multilevel-example1
  class voice
    priority level 1
    police cir 5000000 3125
  class video
    priority level 2
    police cir 10000000 12500
```

Note If you do not explicitly configure a level, a priority queue will operate as a P1 queue. However, if you want to configure multilevel priority queuing, you must explicitly configure levels.

For example, the following configuration would be rejected - you need to explicitly configure the priority level in the voice class:

```
policy-map multilevel-rejection-example
  class voice
    priority
    police cir 5000000 3125
  class video
    priority level 2
    police cir 10000000 12500
```

Bandwidth Queues

Bandwidth queues enable you to apportion interface bandwidth for applications that lack strict latency requirements. Recall that the intent of scheduling is to ensure that all applications receive the necessary bandwidth and to utilize unused bandwidth by making it available for other applications.

You can reflect on *bandwidth sharing* as follows:

Guarantee bandwidth for applications so that they operate effectively. For example, you may decide that your email application is business critical and must continue to operate even during network congestion. If so, you would want to always guarantee some amount of available bandwidth to your business critical applications.

Determine which applications to sacrifice under congestion. For instance, you may decide that a social media application is not business critical; employees can use the network for such applications but not at the expense of business critical activities. If so, you can place these applications in a queue that is intentionally deprived of service during congestion.

As described in How Schedule Entries are Programmed, on page 3, bandwidth queue schedule entries have three distinct parameters Min, Max, and Ex set by **bandwidth**, **shape**, and **bandwidth remaining** commands, respectively. Let's take a closer look at these commands.

Bandwidth Command

The **bandwidth** command sets the Minimum bandwidth (Min) guarantee in a schedule entry at which a queue will be serviced. Given the exact bandwidth requirements of an application, this command provides a convenient way to ensure that an application receives exactly what it needs under congestion. Be aware that by default every entry will also have a configured Excess Weight, which can lead to some additional guaranteed service for the queue.

Bandwidth guarantees are configured in Kbit/sec and may be configured in increments of 1 Kbps. You can also configure the guarantee as a percentage of physical linerate: a percentage of the nominal interface rate displayed as bandwidth through the **show interface** command.

For example, the **show interface gigabit x/y/z** command on a GigabitEthernet interface would show a BW of 1000000 Kbit/sec and any percentage value would be a percentage of this nominal rate. So, if we configured **bandwidth percent 50** on a Gigabitethernet interface, it would set a Min value of 500 Mbps.

The **bandwidth** command accepts the **account** keyword, which enables you to adjust what overhead is included in rate conformance calculations. However, any configured **account** value must be consistent across a policy-map (all **bandwidth** and all **shape** commands in the policy-map must be configured with the same account value).

Caution Do no configure extremely low bandwidth guarantees on high speed interfaces.

Recall from How Schedule Entries are Programmed, on page 3, that we service queues with Min bandwidth guarantees before Excess queues. Furthermore, recall from What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 9 that by default the scheduling length of a packet does not include all of the physical bandwidth that will be consumed to transport that packet (it does not include CRC or inter packet overhead). Given these two facts you should be careful not to guarantee more bandwidth than is actually available. Else, you may starve queues possessing only an excess weight of service.

Imagine that you configure a queue with a Min of 98 Mbps and attach the policy to a FastEthernet interface (100 Mbps). If we send all 100 byte frames, the *scheduling length* for each frame would be 96 bytes but the *actual bandwidth* consumed by each (including the required inter- packet overhead) would be 120 bytes.

From a scheduling length perspective, 98 Mbs would translate to 120 Bytes/96 Bytes * 98 Mbps = 122.5 Mbps of physical bandwidth usage:

actual bandwidth/scheduling length * Min = physical bandwidth usage

(100 bytes Frame + 20 bytes Per Packet Overhead)/(100 bytes Frame - 4 bytes CRC) * 98 Mbps = 120 bytes/96 bytes * 98 Mbps = 122.5 Mbps

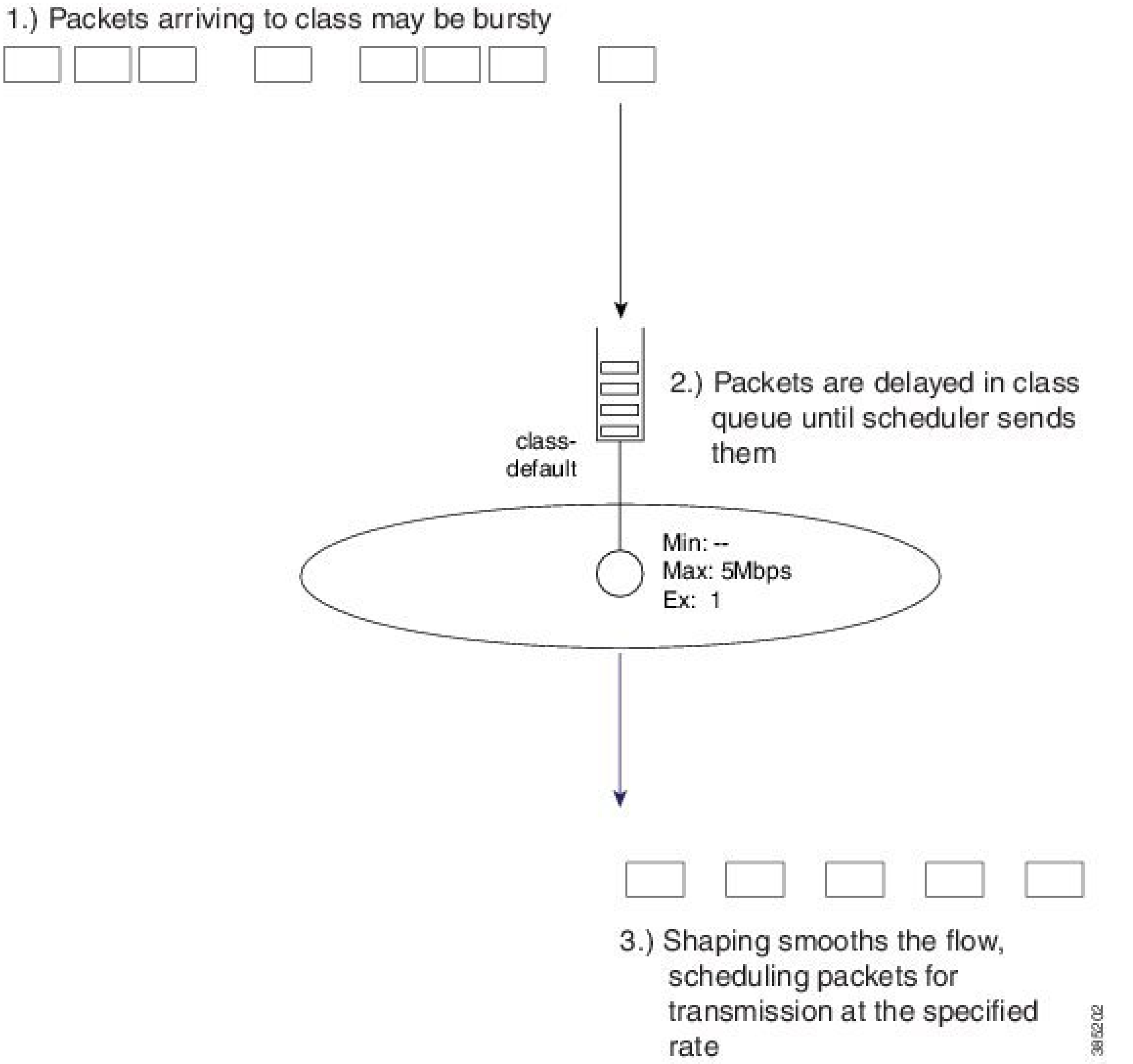
So, we cannot honor our promise! Generally, if the sum of your priority guarantees and Min bandwidth guarantees totals 75% or more of the physical bandwidth, consider whether you are starving other queues of service. In particular consider what might happen to class-default traffic as the default configuration for the class-default schedule entry is to configure an excess weight only.

Shape Command

The **shape** command sets the Max rate in a schedule entry at which a queue will be serviced. Setting the Max rate does not in itself guarantee any throughput to that queue; it simply sets a ceiling. If you create a class containing just the **shape** command it will also receive the default excess weight setting (‘1'), which determines the bandwidth share that class should receive.

Shaping is most commonly used in hierarchical policies. Occasionally, however, you might want to use shaping in flat policies. That is, you may have adaptive video in a bandwidth class that if unconstrained could expand its bandwidth usage to beyond what is physically available. Consequently, you might want to employ shaping to limit the expansion of that flow.

Figure 11: Single-shaped Queue


1.) Packets arriving to class may be bursty
2.) Packets are delayed in class queue until scheduler sends them
class-default
Min: --
Max: 5Mbps
Ex: 1
3.) Shaping smooths the flow, scheduling packets for transmission at the specified rate
385202

The diagram above shows an example of a single-shaped queue. For this simple example, the configuration would look as follows:

```
policy-map shape_example
  class class-default
    shape average 5m
```

As packets arrive they are added to the end of the queue for that class. The scheduler is pulling packets from the head of the queue at the specified rate. If the *arrival rate* (rate at which packets are arriving at the queue) exceeds the *service rate* (the rate at which packets are pulled from the queue) then packets will be delayed and must sit in the queue until all preceding packets are sent. In this simple example no other queues compete for bandwidth, so the service rate will equal the shape rate (5 Mbps).

From this simple example you can see that a shaper will "smooth" a stream. Typically, it will be a few small packets rather than a single packet released by the scheduler. The net result is as shown, a shaper meters the rate at which packets are forwarded.

Shape Average

The **shape average** command is the primary means of configuring a Max rate for a class.

You can configure the rate in bits per second or as a percentage of the interface (or parent shaper) rate. As with other scheduling commands, you can adjust the overhead included in scheduling calculations with the **account** keyword.

As an example, we can modify the previous configuration snippet to include CRC and inter-packet overhead on an Ethernet interface as follows (for more details see What's Included in Scheduling Rate Calculations (Overhead Accounting), on page 9):

```
policy-map shape_example
  class class-default
    shape average 5m account user-defined 24
```

Note The **shape average** command-line interface also includes options for Bc (bits per interval, sustained or committed) and Be (bits per interval, excess). (These options are remnants from the software implementation of shaping in IOS classic and have no effect on an ASR 1000 Series Aggregation Services Routers.)

On software implementations the processing overhead meant it was only feasible to perform the math involved in scheduling at some predetermined interval, typically a number of milliseconds.

Adjusting Bc was a way to further reduce scheduling frequency (and thereby processing overhead) at the expense of more burstiness in forwarded traffic. On the ASR 1000 Series Aggregation Services Routers, scheduling decisions are performed in dedicated hardware and (**so?**) frequent scheduling decisions does not incur a performance penalty. We have optimized the hardware to maximize the elimination of burstiness from the stream it forwards, obviating user input on Bc or Be.

Shape Peak

The **shape peak** command is supported on the ASR 1000 Series Aggregation Services Router but it offers no functionality beyond the **shape average** command. We support it to easily migrate configurations from existing IOS classic devices to ASR 1000 Series Aggregation Services Routers. With **shape peak** command, the router will look at the configured rate, Bc and Be and then calculate a target shape rate. This rate displays in the **show policy-map interface** command output and on the ASR 1000 Series Aggregation Services Router is programmed into the hardware schedule entry. If you are creating a new configuration, you should use the **shape average** command.

Bandwidth Remaining Command

The **bandwidth remaining** command configures the excess weight in a schedule entry and so determines a queues share of the excess bandwidth. Recall that excess bandwidth is defined as any bandwidth that is neither explicitly guaranteed to another queue by the **priority** or **bandwidth** command nor used by a queue to which it is guaranteed. (For details on excess weight, see How Schedule Entries are Programmed, on page 3.) By distributing excess bandwidth sharing in a deterministic manner (behavior entirely determined by initial state), we avoid wasting bandwidth. (For further discussion of bandwidth sharing, see Bandwidth Queues, on page 22.)

The **bandwidth remaining** command is also an effective way to guarantee bandwidth to queues. It is perfectly reasonable, and very common, to apportion all bandwidth using only excess bandwidth sharing.

The **bandwidth remaining** command has two variants - **bandwidth remaining ratio** and **bandwidth remaining percent**. in either case, you are setting the same excess bandwidth parameter in a schedule entry. The rationale for two forms will make sense when we discuss hierarchical policies. In the context of a flat policy attached to a physical interface, however, you can choose whichever form simplifies provisioning.

Note Both variants (as similar to other scheduling commands) support the **account** keyword.

Bandwidth Remaining Ratio

Concerning the **bandwidth remaining ratio** command, the first thing you need to understand is that every bandwidth queue schedule entry will have a default Excess Weight (Ex) of one ('1') provided you do not explicitly set the value. (For example, upon creation, the schedule entry for the class-default queue will have an Ex of 1.) Having a deterministic and easy to understand default removes any ambiguity when designing a QoS scheme.

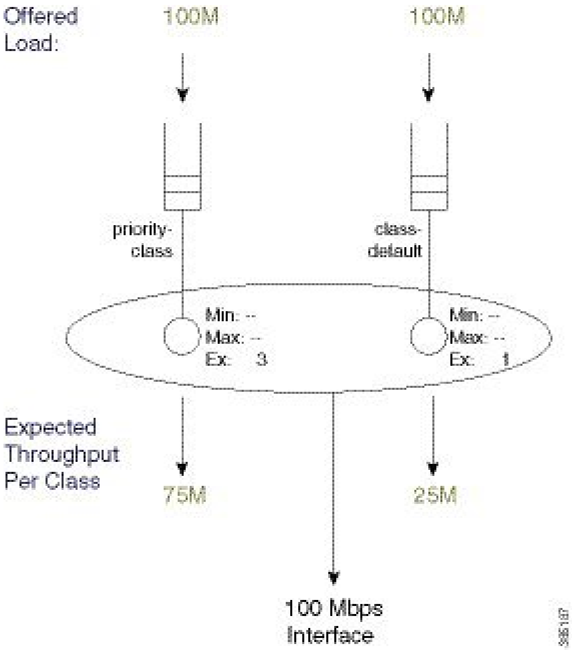
Consider the following policy-map example:

```
policy-map BRR-Example1
  class mission-critical
    bandwidth remaining ratio 3
```

This policy has 2 queues, one for the mission-critical class we explicitly create with a scheduling command and one for the implicit class-default.

Let's now attach this policy to a 100 Mbps interface and offer 100 Mbps to each queue. The scheduling hierarchy and expected throughput per class would be as shown:

Figure 12: Splitting Bandwidth Explicitly Assigned by Ratio



Now let's modify the policy by adding an explicit class with the **shape** command. Recall from the command page for shape (class-map) that the **shape peak** command sets the Max of the schedule entry for that queue. Because Ex has not been explicitly configured, it will default to 1.

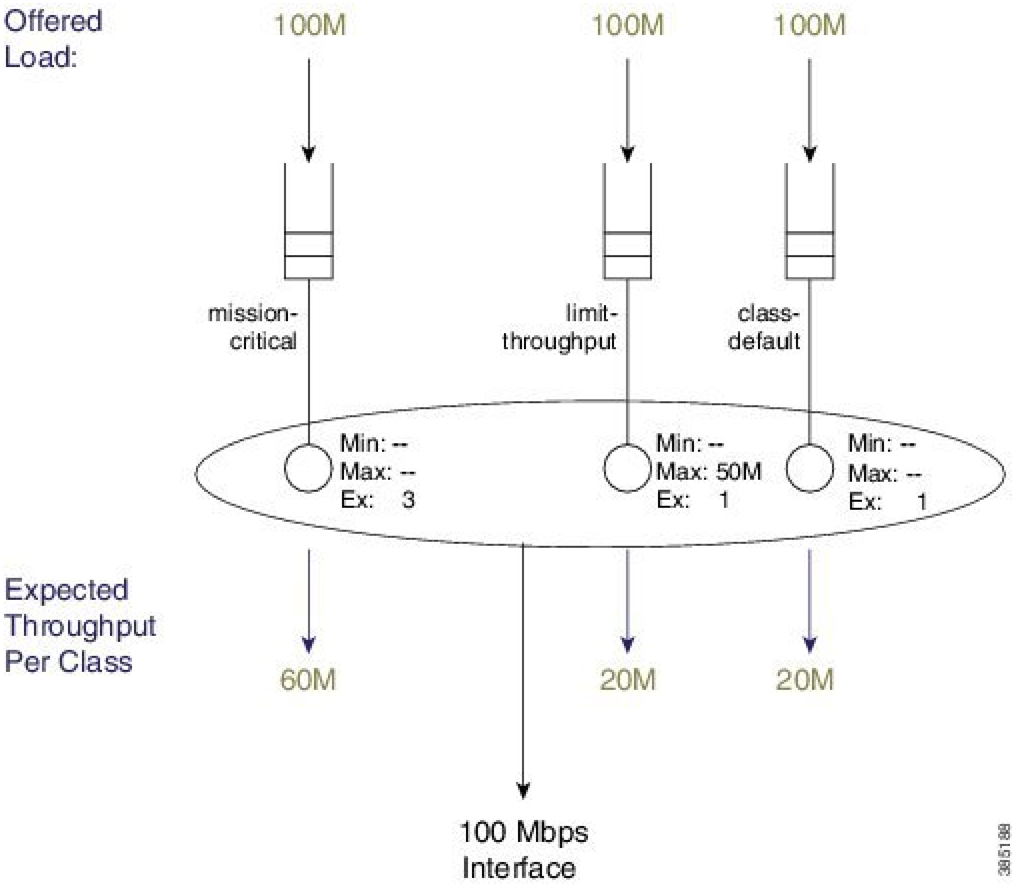
Note A Max entry does not guarantee any share of the bandwidth to a queue, it simply sets a ceiling on the possible throughput for that queue.

The policy could look like this:

```
policy-map BRR-Example1
  class mission-critical
    bandwidth remaining ratio 3
  class limit-throughput
    shape average 50m
```

If we attach this policy to a 100 Mbps interface and offer 100 Mbps to each class, the scheduling hierarchy and expected throughput would look as follows:

Figure 13: Modifying Excess Weight of Explicit Classes with bandwidth remaining ratio Command



The expected throughput (60M, 20M, and 20M) reflects the ratio of Ex values: 3, 1 , and 1. The key point is that modifying the excess weight using the **bandwidth remaining ratio** command will only alter the entry for the class you are explicitly modifying.

The bandwidth remaining ratio ranges from 1 to 1000 so we can achieve considerable variance between the service rate for different queues.

Bandwidth Remaining Percent

The **bandwidth remaining percent** command is another way to modify the Excess Weight (Ex) in a bandwidth queue's schedule entry. Obviously, with a percent-based scheme, the sum of excess weights across all bandwidth queues must total 100. We achieve this by distributing (equally) any percentage (not explicitly assigned) across class-default and any other queues that are not configured explicitly.

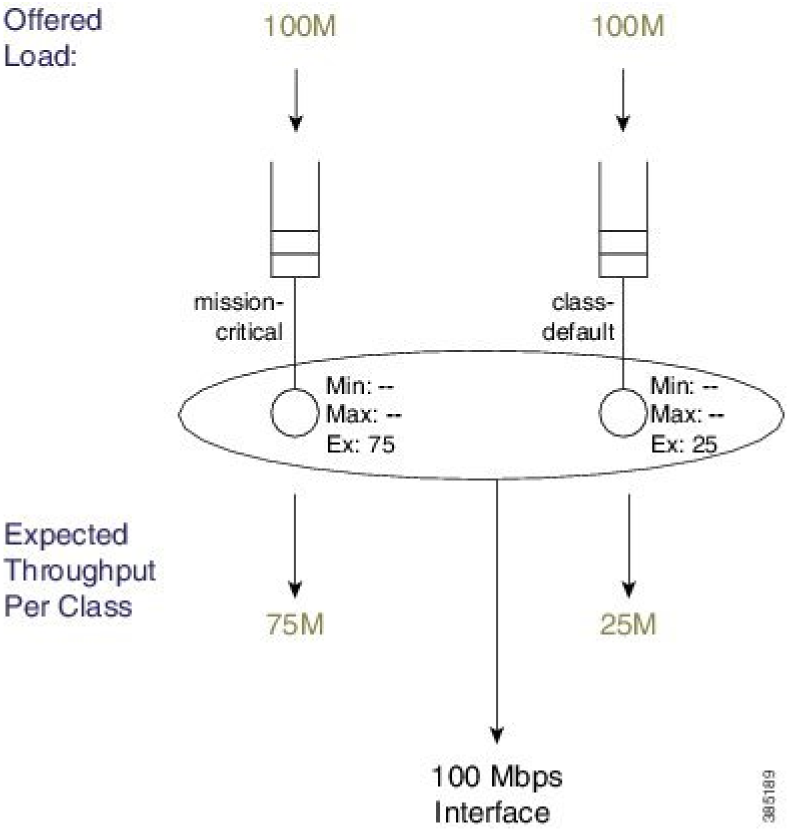
For details on this command, refer to the command page for bandwidth [remaining percent].

Consider the simplest example, which is equivalent to the first example in Bandwidth Remaining Ratio, on page 27:

```
policy-map BRP-Example1
  class mission-critical
    bandwidth remaining percent 75
```

The scheduling hierarchy and expected throughput per class will look as follows:

Figure 14: Splitting Bandwidth Explicitly Assigned by Percent



Notice how the Ex of class-default was changed (from "1," by default) even though it was not explicitly configured.

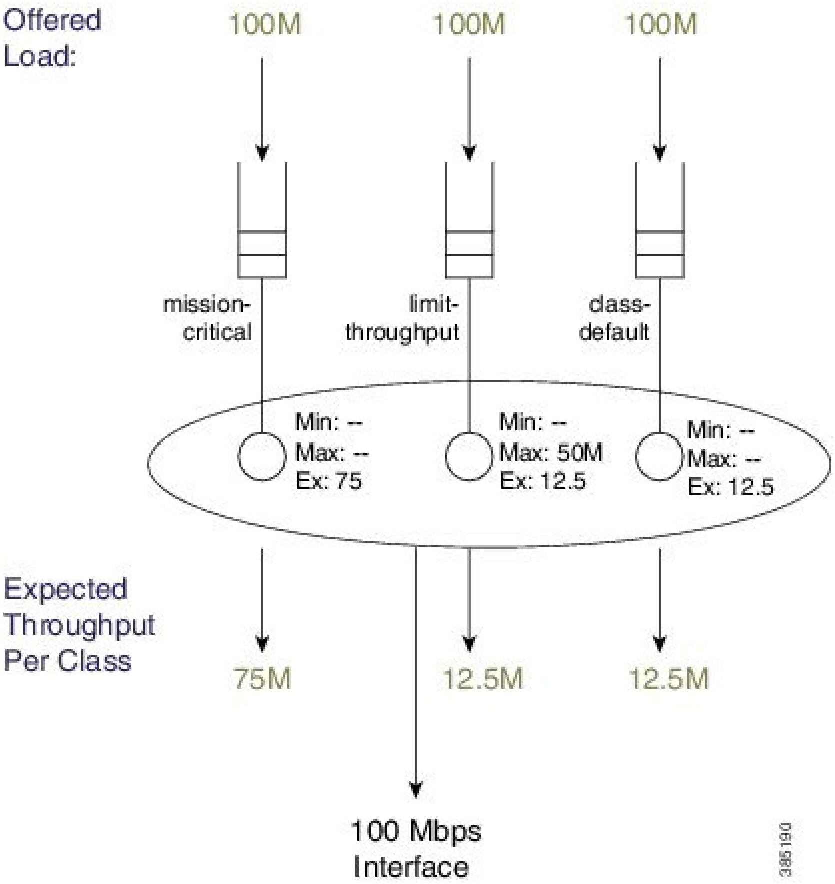
Now let's add a queuing class with no explicit bandwidth remaining configuration - again we'll add a class with just a shaper (see the figure "Splitting Bandwidth Explicitly Assigned by Ratio" in Bandwidth Remaining Ratio, on page 27):

```
policy-map BRP-Example2
  class mission-critical
    bandwidth remaining percent 75
  class limit-throughput
    shape average 50m
```

This example highlights the behavior of splitting percentage across class-default and any classes that are not explicitly assigned.

The hierarchy and throughput will now look as follows:

Figure 15: Splitting Bandwidth Percentage Across class-default and Unassigned Classes with an Added Shaper



Two-Parameter versus Three-Parameter Scheduling

Earlier we described how the schedule entry for each bandwidth queue has three parameters to control queue service: Min, Max and Ex. (See How Schedule Entries are Programmed, on page 3.) This is why we categorize the scheduler implementation on the ASR 1000 Series Aggregation Services Router as a *three-parameter scheduler*.

In an existing IOS classic implementation, we provide a simpler *two-parameter scheduler*. Instead of distinct entries for Min and Ex, each schedule entry has only a single weight. Whether you used the **bandwidth** or **bandwidth remaining** command you were configuring the same single weight. To grasp the difference, let's look at an example focusing on the **bandwidth** command.

The policy-map for this example will look as follows:

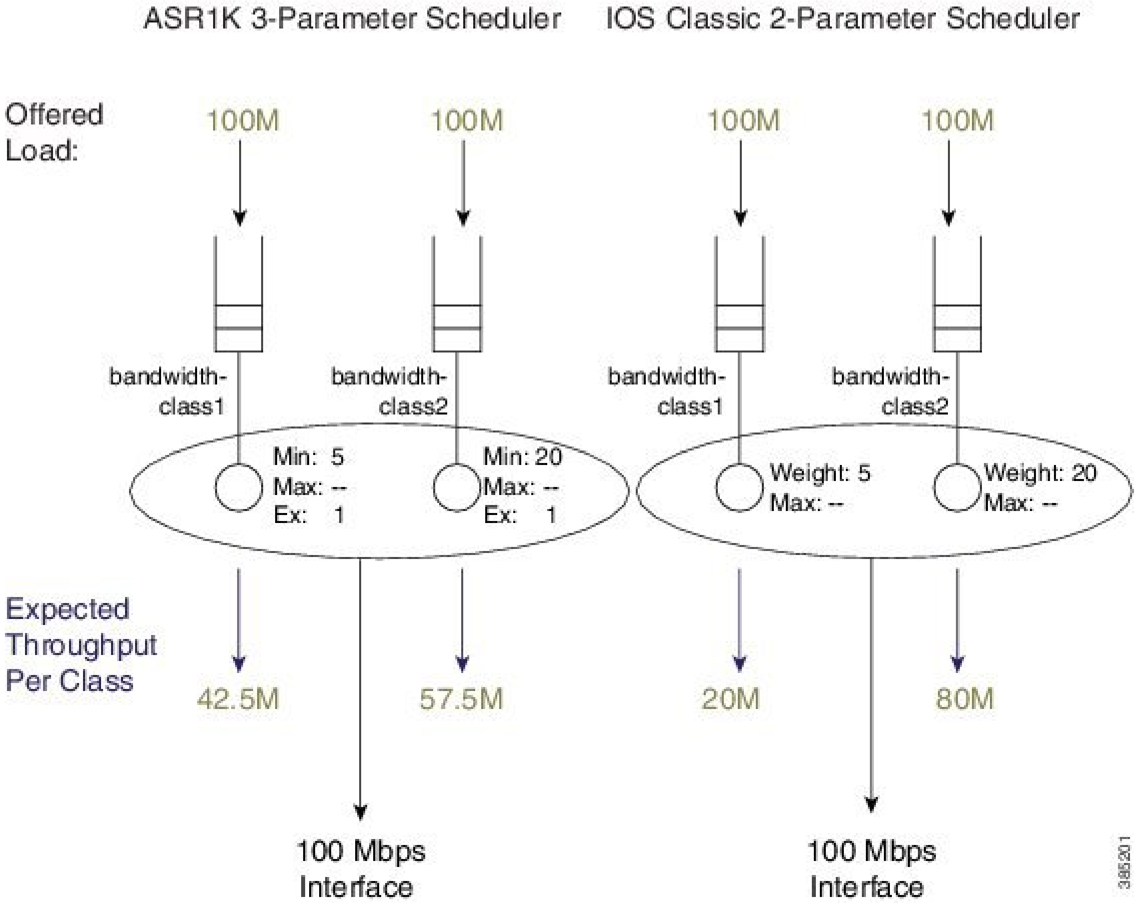
```
policy-map bandwidth-example
  class bandwidth-class1
    bandwidth percent 5
  class bandwidth-class2
    bandwidth percent 20
```

Now consider the policy-map attached to a 100 Mbps interface and offer 100 Mbps to each queue. The following figure shows how the schedule configuration and expected throughput would appear both on an ASR 1000 Series Aggregation Services Router (on the left) and on a router running an IOS Classic image (e.g., a Cisco 7200; on the right).

25 Mbps is assigned to the ASR 1000 Series Aggregation Services Router three-parameter scheduler and we use it to honor Min guarantees, which leaves 75 Mbps of excess bandwidth. This excess is shared equally between the two queues based on the default excess-weight of '1' that each queue will receive.

If we apply the same configuration on a two-parameter scheduler, the configured bandwidth values will dictate a single-weight parameter in the schedule entry. The concept "Min scheduling versus excess bandwidth sharing" does not apply here. Instead, for each entry, all bandwidth sharing hinges on the single weight.

Figure 16: Same Configuration running on ASR 1000 with IOS XE and Router running IOS Classic



Looking at this example you see that the same configuration on an ASR 1000 Series Aggregation Services Router running IOS XE can yield significantly different behavior from a router running an IOS classic image.

Note To achieve identical behavior to an router running IOS Classic, you can use a configuration using only excess bandwidth sharing. Changing **bandwidth percent** statements in an IOS Classic configuration to **bandwidth remaining percent** statements in IOS XE is an easy way to migrate existing configurations.

Schedule Burstiness

Possible sources of burstiness in scheduling include: packet batching and the scheduler's representation of time.

Packet Batching

This source is intentional and should not cause concern. As implemented in hardware, we cap the number of decisions a schedule can make per second. If you were to send all small packets, say 64-byte frames, a schedule may struggle to maintain if it is making decisions, packet by packet, for a fast interface like 10 Gbps. To ease this burden, the hardware will batch small packets (up to about 512 bytes from the same queue) and let the scheduler treat them as a single decision.

So, if a single packet of 512 bytes were at the head of the queue we would send that to the schedule as a single packet. On the contrary, if five 64-byte packets were at the head of the queue we would batch the packets as a single packet from the scheduler's perspective. That is, we would pull all five packets from the queue simultaneously and forward them back to back on the wire as a single burst. As the size of a single MTU greatly exceeds that of a burst, the later negligibly impacts downstream buffering or jitter for other queues.

Scheduler's Representation of Time

The second potential source of burstiness arises from how a schedule in hardware tracks time. If you mix very small rates (say 100K and less) and very large rates (say 100M and higher) in the same policy-map you may experience unexpected burstiness in scheduling of traffic from the queue configured with the high rate.

When you use *real-time scheduling* (using either the **bandwidth** or **shape** command) you are specifying rates in bits per second. This means that each schedule entry must have a concept of real time and must be monitoring service rate vs that real time. The representation of time must be uniform across all entries in a given scheduler.

Consider an 8-Kbps shaper (8000 bits/sec = 1000 bytes/sec).

> Sending 64-byte packets would be (equivalent to) sending one packet every (64-byte packet * 64/1000 =) 64 mS.
>
> Sending 1500-byte packets would be (equivalent to) sending one packet every 1500 byte * 1500/1000 =) 1.5 sec.

We want to represent anything ranging from 64 mS to many seconds. To do this we count time and establish that every counter increment represents 10.5 mS of real time.

Now consider a 10-Gbps shaper. In 10.5 mS we would expect to send 8,750 1500-byte packets:

> 10,000,000,000 b/sec * .0105 sec = 105,000,000 bits, which equals 105,000,000/8/1500, or 8750 1500-byte packets

This is a huge data burst. If we were counting time in 10.5 mS increments, whenever the clock (advances) we would need to send that burst. In contrast, if .65 mS represented *real time*, we would expect to send 542 1500-byte packets (a far more manageable situation).

The representation of time is driven by the lowest rate configurable within a policy-map. The following table shows the granularity of time chosen vs. rate that is configured in a policy-map. (The details are accurate for ESP-20 but only similar for all variants of ASR1K hardware.)

Table 2: Granularity of Time Chosen vs Configured Rate

Rate Range in Policy-Map	Chosen Time Granularity
8K - 14K	10.5 mS
15K - 28K	5.2 mS
29K - 57K	2.6 mS
58K - 115K	1.3 mS
116K - 231K	0.65 mS
232K - 463K	0.33 mS
464K - 927K	0.16 mS
928K - 3094K	0.08 mS
3.1M - 6.1M	40 uS
6.2M - 12.2M	20 uS
12.3M - 24.6M	10 uS
24.7M - 49.4M	5 uS
49.5M - 99M	2.5 uS
99.1M - 198M	1.3 uS
198.1M - 396M	0.6 uS
396.1M - 10G	0.3 uS

Reading the table you observe that if all rates in your policy-map are 116K or greater, then any burst introduced by this representation of time would last less than a millisecond and therefore insignificant. If you configure shape or bandwidth rates less than 116K on a fast interface you may want to ensure there are no unintended consequences. (e.g., if all rates in your policy-map range from 29K - 57K, then any burst introduced by this representation of time would be 2.6 mS in duration!) Such consequences could include downstream devices dropping if they have insufficient buffering to receive such a burst, WRED dropping packets, or downstream policers dropping packets due to exceeding their burst allowance.

Minimum Guaranteed Service Rate for a Queue

For any queue you can calculate a minimum guaranteed service rate (the service that a queue will receive if all other queues are congested). In some prior examples we have shown an offered rate of 100% to each queue - the expected throughput in those examples is the minimum guaranteed service rate. Knowing this rate might inform you on how your applications will behave under severe congestion. That is, when the network is systemically overloaded, can you expect your application to run?

One particularly useful number you can calculate from the guaranteed rate is the delay a packet would experience if the egress queue were full. For example, let's consider an oversubscribed video queue, whose policy-map looked as follows:

```
policy-map min-service-rate-example
  class priority-class
    priority
    police cir 1m 1250
  class video
    bandwidth 1000
  class mission-critical
    bandwidth 2000
```

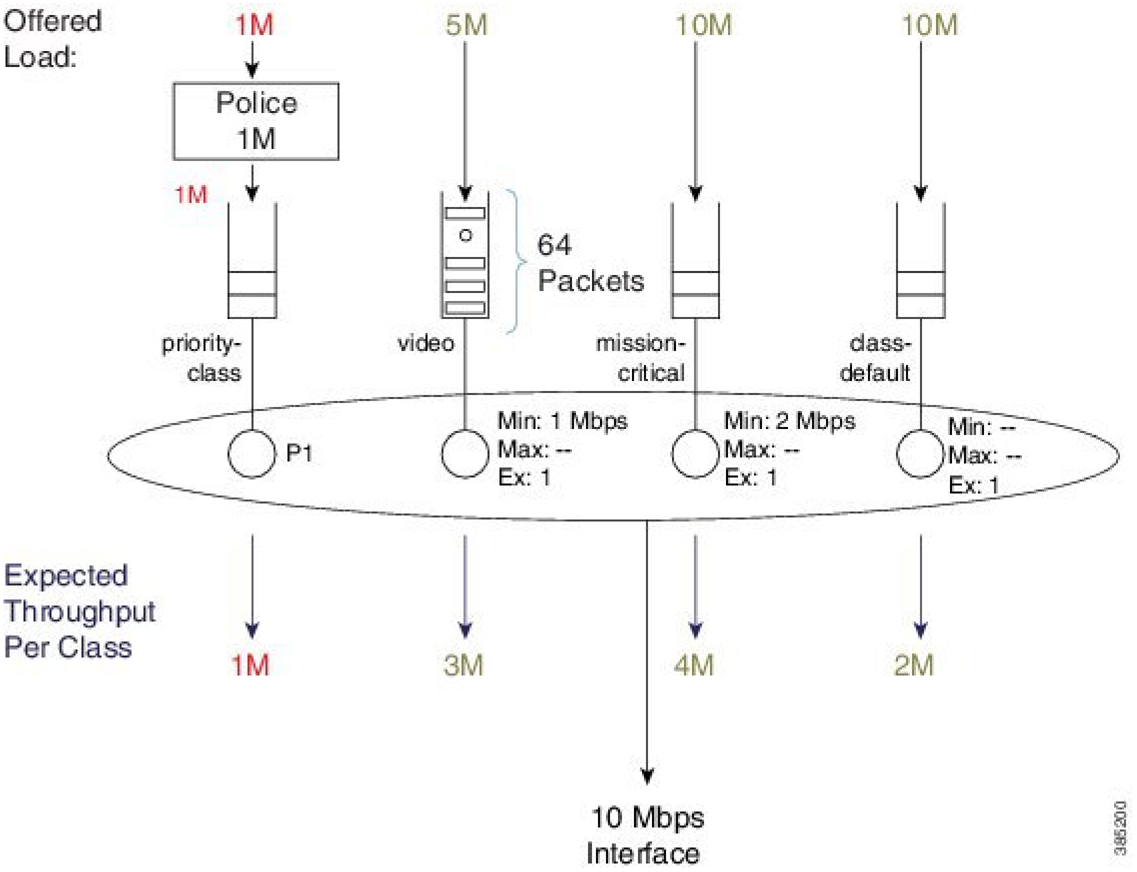
In this example we attach the policy-map to a 10 Mbps Ethernet interface and the offered load to each class as shown in the following analysis. (We will abide by the guidelines in Schedule Operation, on page 4.)

The video class is serviced at its minimum guaranteed service rate of 3 Mbps (1 Mbps (Min configured with the **bandwidth** command) + 2 Mbps (its always guaranteed share of excess bandwidth). Looking more closely at the calculations involved:

10 Mbps - 1 Mbps(for P1) - 3 Mbps(Min guaranteed for video and mission-critical) = 6 Mbps

6 Mbps excess bandwidth(after accounting for P1 and Min guarantees)/3 equal shares for all queues = 2 Mbps

Figure 17: Minimum Service Rate and "Experience" of Latency



With the minimum guaranteed service rate, you can now calculate "the experience" of latency packets arriving at the video queue. Using the default queue-limit of 64 packets (under over-subscription, we would expect the queue to fill and contain 64 packets) a new packet would be either dropped or placed at the tail of the queue (if the packet arrived just when another was pulled from the video queue).

Given this queue for video traffic, we could expect the average packet size to be about 1400 bytes (roughly the size of an MPEG I-frame), generating 716,800 bits as the amount of data buffered and awaiting transmission:

64 packets * 1400 bytes/packet * 8 bits/byte = 716,800 bits

Given a minimum rate of 3 Mbps, we would require 239 mS to drain this queue:

716,800 bits / 3 Mbps = 0.239 Seconds (239 mS)

As you can see, a minimum guaranteed service rate can help you envisage (and so predict) the behavior of your applications under congested conditions.

Pak Priority

Pak priority designates a scheme to protect some critically important control packets (interface keepalives, BFD packets, some routing protocol hellos etc.) that are vital for network stability. In this section, we will describe these packets and outline how they are scheduled.

Note The name, pak_priority, is unfortunate and might cause confusion because the control packets are not (actually) queued in a priority queue.

With pak_priority, we attempt to ensure guaranteed delivery and not guaranteed low latency) of the control packets. They are marked with an internal pak_priority flag, when first generated in the control plane. This flag does not propagate outside the router and is only used to ensure we give special treatment to the packet as we send it towards the egress interface.

Observe that we set the DSCP of IP encapsulated control packets to CS6 protecting them at other devices in the network where they must traverse. On the router that generates the control packet, the pak_priority designation dictates further protection beyond what you might configure for CS6 packets.

The following table lists the packets and protocols we mark with the internal pak_priority flag.

Packets and Protocols Marked with the pak_priority Flag

Table 3: Control Packets Marked with pak_priority Flag

Level Marked	Packets and Protocols
Layers 1 and 2	
	ATM Address Resolution Protocol Negative Acknowledgment (ARP NAK)
	ATM ARP requests
	ATM host ping operations, administration and management cell(OA&M)
	ATM Interim Local Management Interface (ILMI)
	ATM OA&M
	ATM ARP reply
	Cisco Discovery Protocol
	Dynamic Trunking Protocol (DTP)
	Ethernet loopback packet

Level Marked	Packets and Protocols
	Frame Relay End2End Keepalive
	Frame Relay inverse ARP
	Frame Relay Link Access Procedure (LAPF)
	Frame Relay Local Management Interface (LMI)
	Hot standby Connection-to-Connection Control packets (HCCP)
	High-Level Data Link Control (HDLC) keep-alives
	Link Aggregation Control Protocol (LACP) (802.3ad)
	Port Aggregation Protocol (PAgP)
	PPP keep-alives
	Link Control Protocol (LCP) Messages
	PPP LZS-DCP
	Serial Line Address Resolution Protocol (SLARP)
	Some Multilink Point-to-Point Protocol (MLPP) control packets (LCP)
IPv4 Layer 3	
	Protocol Independent Multicast (PIM) hellos
	Interior Gateway Routing Protocol (IGRP) hellos
	OSPF hellos
	EIGRP hellos
	Intermediate System-to-Intermediate System (IS-IS) hellos, complete sequence number PDU (CSNP), PSNP, and label switched paths (LSPs)
	ESIS hellos
	Triggered Routing Information Protocol (RIP) Ack
	TDP and LDP hellos
	Resource Reservation Protocol (RSVP)

Level Marked	Packets and Protocols
	Some L2TP control packets
	Some L2F control packets
	GRE IP Keepalive
	IGRP CLNS
	Bidirectional Forwarding Protocol (BFD)

Levels of Protection for pak_priority Packets

First level

A policer or WRED will not drop packets with this designation.

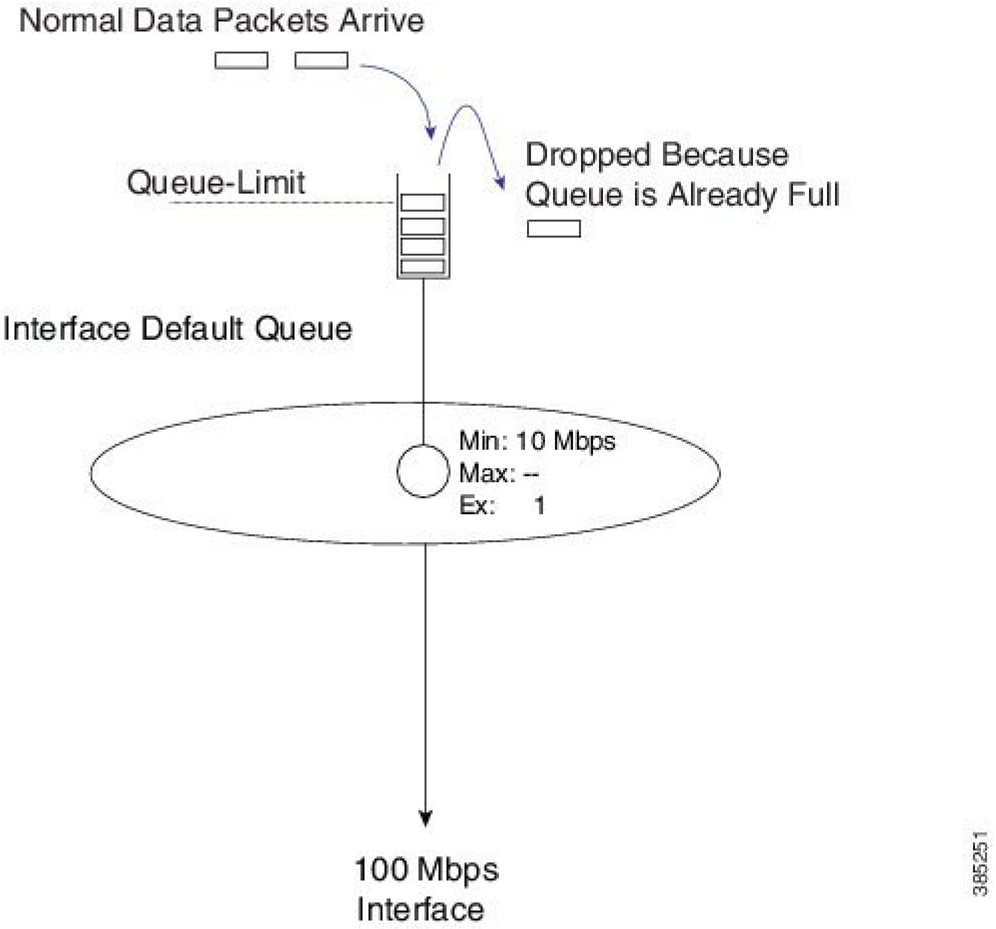
Second level

We will enqueue pak_priority packets even if an egress queue is already full. To grasp this let's first look at a physical interface that has no QoS configured, where we still need a queue and (therefore) a schedule to pull packets from that queue.

Without QoS configured on the interface, we have a single first-in first-out (FIFO) queue (referred to as the *interface default queue*). (Do not confuse with a *class-default queue*).

In the diagram below, normal data packets arrive when the queue is full. Notice that the schedule entry has the typical Min and default Ex values (10% of the interface rate and 1). Because only one queue exists, these values have no effect; without competition, one queue will receive all the available bandwidth.

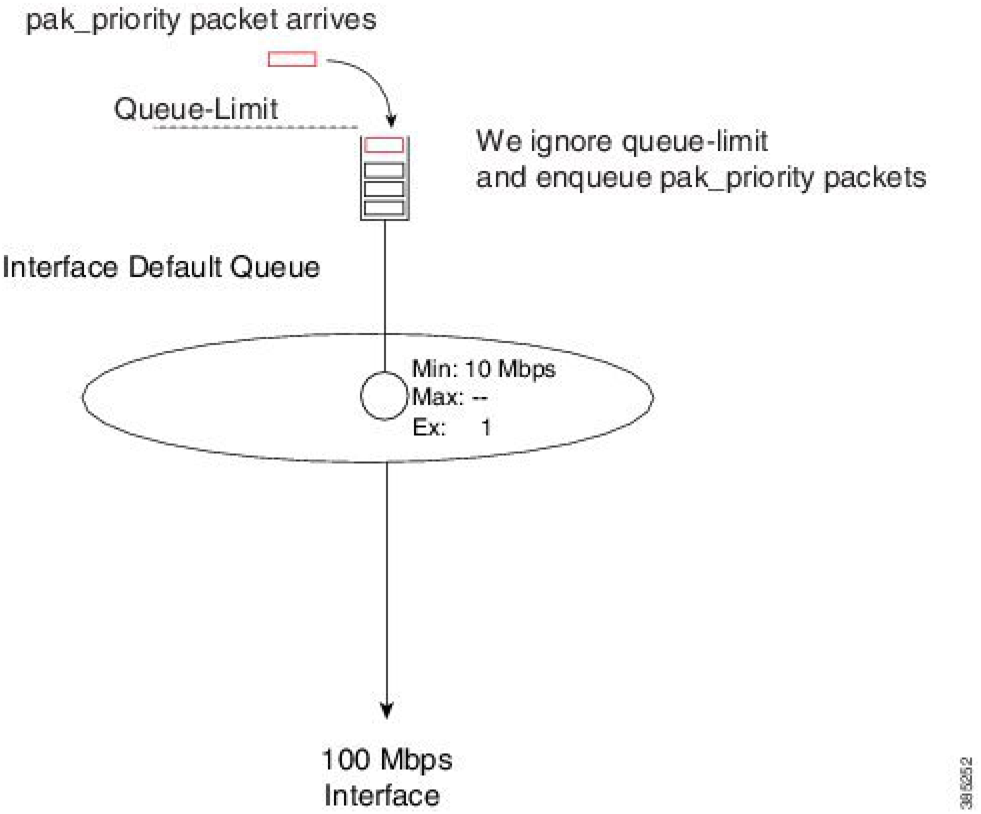
Figure 18: Normal Packet Arrives when Queue is Full



As with every queue, a queue-limit determines how much data we can buffer before the queue is considered full.

If a pak_priority packet arrives while the queue is full, we ignore the queue-limit and enqueue it; the packet must wait until all leading packets are sent. On most ASR 1000 Series Aggregation Services Router platforms, the default queue-limit is 50 mS. So, we may delay the pak_priority packet for 50 mS but we do guarantee its delivery.

Figure 19: pak_priority Packet Arrives when Queue is Full, and we Ignore queue-limit

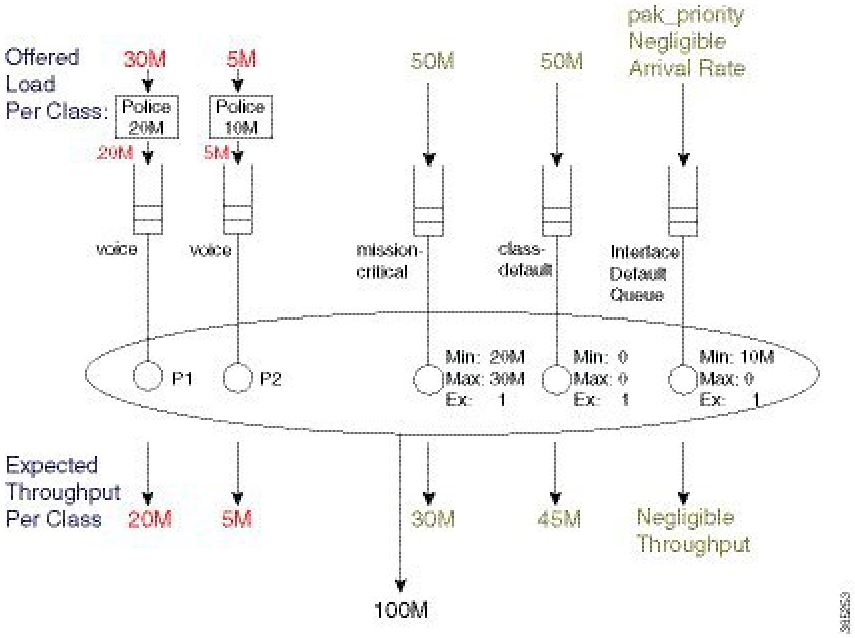


The discussion (of pak priority) changes slightly when you configure QoS on an interface. To illustrate, let's reconsider one of the very first examples in this chapter (see Schedule Operation: With a Shaper, on page 7).

```
policy-map scheduling-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
    shape average 30m
```

We didn't show it previously and as you will now observe below, when you configure a policy we do not remove the interface default queue.

Figure 20: Repurposing Interface Default Queue as a Dedicated pak_priority Queue



Pak_priority packets are still enqueued in the interface default queue. Essentially, the queue has been repurposed as a dedicated pak_priority queue.

Looking at the diagram you can now see the importance of the Min value (set at 10% of linerate). This value ensures that other queues (with a Min service configured) cannot deprive this queue of service.

However, although we configure a Min of 10% of linerate we will never consume this bandwidth. We only mark a very small number of critical packets as pak_priority so the rate at which they arrive is negligible. We overpromise on the Min with the understanding that it will not impact behavior of other queues. Were we to mark too many packets "pak_priority," this scheme would not work.

With routing protocols we mark Hello packets but not routing updates with pak_priority. So, you should create a bandwidth queue for CS6 packets.

The Hello packets will pass through the interface default queue and the routing updates will use the newly-created bandwidth queue.

An exception is BGP, where we do not mark Hello packets as pak_priority. Why? BGP Hello packets and updates share the same TCP stream. Providing special treatment would cause TCP packets to arrive out of sequence. This offers no benefit as you could not consume them.

Classifying a pak_priority packet (by matching DSCP or any other field) and attempting to move it to a bandwidth queue will not happen - the packet will still be enqueued in the interface default queue. However, you can classify the packet and move it to a designated priority queue.

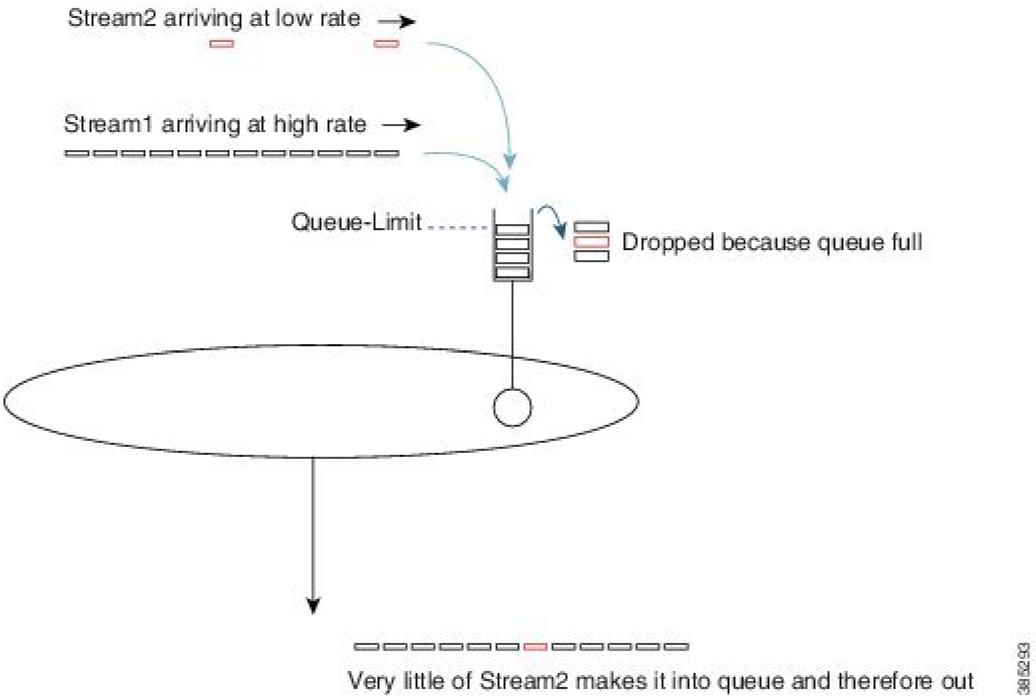
Flow-Based Fair Queuing

Flow-based fair queuing enables us to provide some fairness between flows that belong to the same class in a policy-map. It provides some protection for low speed (well behaved) streams of traffic competing with high-rate streams.

If an individual class is oversubscribed (the offered rate exceeds the service rate), one high-rate flow can starve low-rate flow(s) of service. To understand this you need to consider multiple streams targeting the same physical queue. After it fills, additional packets are dropped. Whenever the scheduler sends a packet from the queue, a single space will open at the queue's tail. We successfully enqueue whichever packet arrives next. If you look at the following example, you notice that a packet from Stream1 (the high rate stream) is more likely to arrive next. Fairness between treatment of the streams is absent! Whatever exits the interface is purely driven by what packet manages to make it into the queue.

Not only does the high rate stream obtain an unfair share of the class bandwidth, it also impacts latency for low rate streams. As successfully-enqueued packets are always at the tail of a full queue, they must wait for all other packets to drain before they can be transmitted.

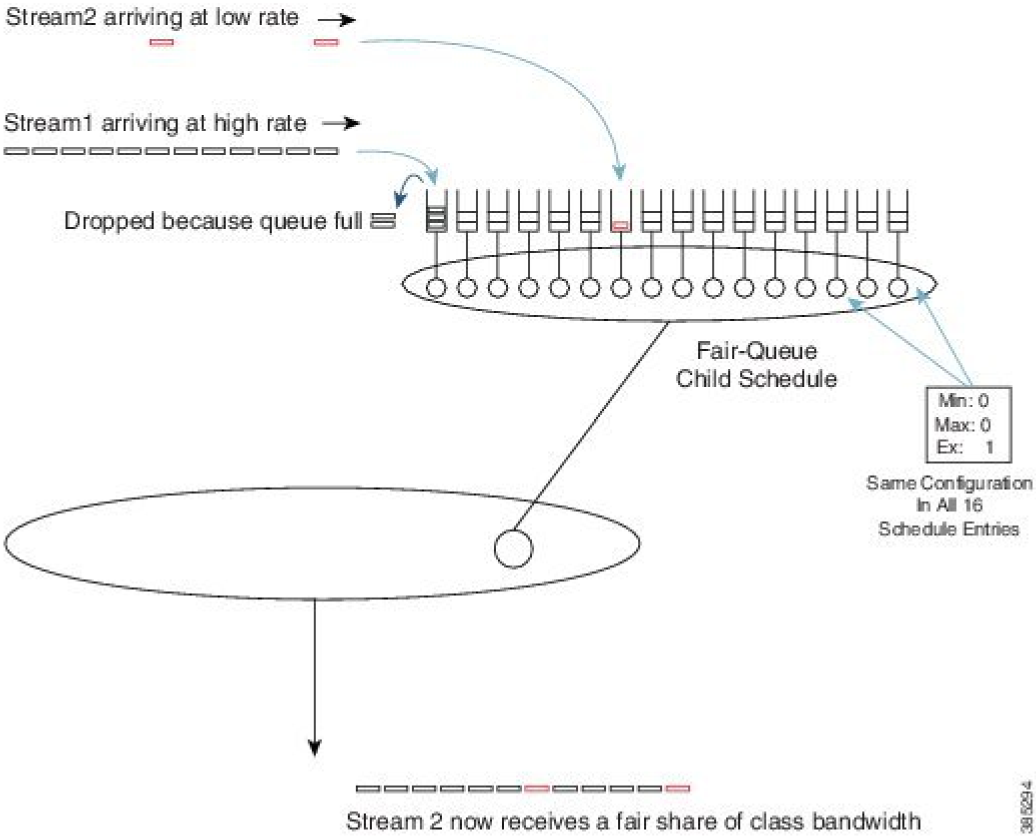
Figure 21: Latency without Flow-Based Fair Queuing


Stream2 arriving at low rate
Stream1 arriving at high rate
Queue-Limit
Dropped because queue full
Very little of Stream2 makes it into queue and therefore out
386293

Flow-based fair queuing can alleviate this issue. When you issue the **fair-queue** command you direct the router to create 16 queues for that single class, which represents a simple form of *hierarchical scheduling*. .

Consider the fair-queue child schedule a *pre-sorter*, providing sorting and fairness between multiple streams targeting the same class.

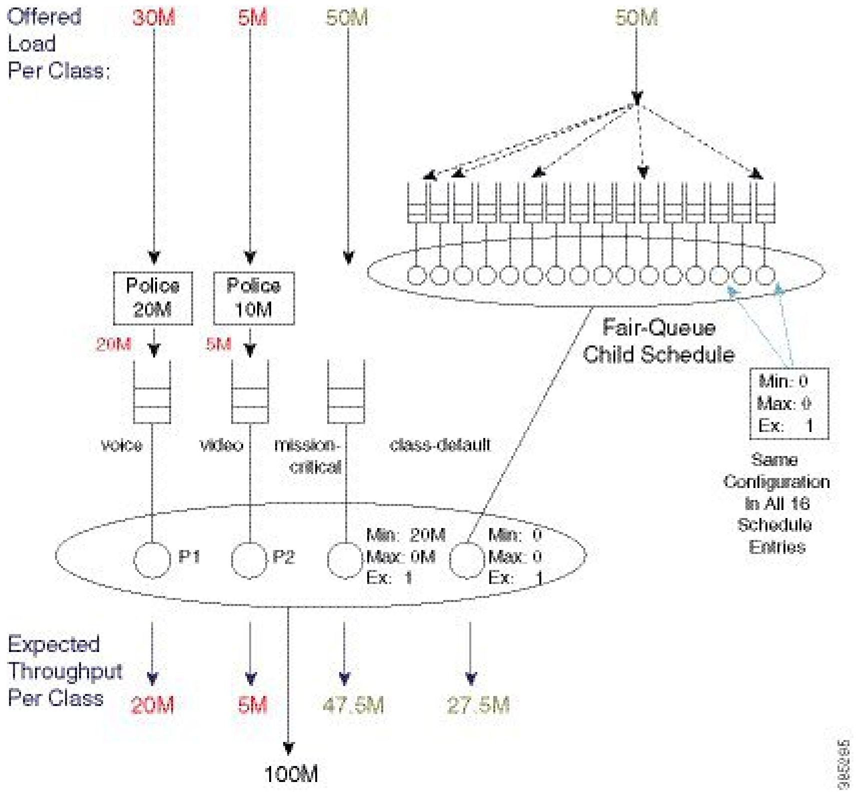
Figure 22: Pre-sorting and Fairness provided by Flow-Based Fair Queuing


Stream2 arriving at low rate
Stream1 arriving at high rate
Dropped because queue full
Fair-Queue
Child Schedule
Min: 0
Max: 0
Ex: 1
Same Configuration
In All 16
Schedule Entries
Stream 2 now receives a fair share of class bandwidth
385294

```
policy-map fair-queue-example
  class voice
    priority level 1
    police 20m
  class video
    priority level 2
    police 10m
  class mission-critical
    bandwidth 20000
  class class-default
```

```
fair-queue
```

Figure 23: Packet Flow with Flow-Based Fair Queuing



Verification

Use the **show policy-map interface** *interface* command to verify operation of scheduling. This command will show long term trends and a complete view of the policy configured.

The data plane sends statistics to the control plane every 10 seconds and control plane refreshs its own statistics every 10 seconds. This means that the values in output of **show policy-map interface** command updates every 10 seconds. Some counters that represent instantaneous state, such as current queue depth, may not be overly useful. It is possible to look directly at hardware counters if you really want true instantaneous state.

The following configuration is an example of **show policy-map interface***interface* command.

```
policy-map show_policy-example
 class voice
  priority level 1
  police cir percent 10 bc 5 ms
 class video
   priority level 2
  police cir percent 20 bc 10 ms
 class critical-data
  bandwidth percent 50
```

This policy has four classes, the three that are explicitly configured and the implicit class-default.

The output from the **show policy-map interface** command mirrors the configured policy. It has a section for each configured class. Within each class the output is consistently organized with a classification section and a section for each configured action.

Note that queuing information for priority classes is shown separately to other features (policers) in that class. This is since multiple priority classes may map into the same queue.

The following is an example of the output from the **show policy-map interface** command. Although it is one continuous block of output we break it into sections to highlight the structure of the output.

<table>
<tr><td><pre>
Device#show policy-map interface g1/0/4
 GigabitEthernet1/0/4

  Service-policy output: show_policy-example

queue stats for all priority classes:
      Queueing
      priority level 1
      queue limit 512 packets
      (queue depth/total drops/no-buffer
drops) 0/0/0
      (pkts output/bytes output)
39012/58518000
</pre></td><td>This section shows queue information for the priority level 1 queue</td></tr>
<tr><td><pre>
    queue stats for all priority classes:
      Queueing
      priority level 2
      queue limit 512 packets
      (queue depth/total drops/no-buffer
drops) 0/0/0
      (pkts output/bytes output)
61122/91683000
</pre></td><td>This section shows queue information for the priority level 2 queue</td></tr>
<tr><td><pre>
Class-map: voice (match-all)
      39012 packets, 58518000 bytes
      5 minute offered rate 672000 bps, drop
 rate 0000 bps
      Match: dscp ef (46)
</pre></td><td>This section shows statistics for the class named voice.
First shown is the classification statistics.</td></tr>
<tr><td><pre>
Priority: Strict, b/w exceed drops: 0
      Priority Level: 1

      police:
          cir 10 %, bc 5
          cir 100000000 bps, bc 62500 bytes

        conformed 39012 packets, 58518000
bytes; actions:
          transmit
       exceeded 0 packets, 0 bytes; actions:

          drop
        conformed 672000 bps, exceeded 0000
 bps
</pre></td><td>Priority level indicates the priority queue above that will be used by this class.
The statistics for the policer used for queue admission control are also shown here.</td></tr>
<tr><td><pre>
Class-map: video (match-all)
      1376985 packets, 2065477500 bytes
     5 minute offered rate 9171000 bps, drop
 rate 0000 bps
      Match: dscp af41 (34)
</pre></td><td>This is start of section for class named video.
Again classification statistics and criteria are shown first</td></tr>
<tr><td></td><td>This section shows actions configured in the class named video.
The statistics for the queue admission control policer.
Priority Level indicates packets from this class will be enqueued in the priority level 2 queue that is shown above.</td></tr>
</table>

police: cir 20 %, bc 10 cir 200000000 bps, bc 250000 bytes conformed 1381399 packets, 2072098500 bytes; actions: transmit exceeded 0 packets, 0 bytes; actions: drop conformed 9288000 bps, exceeded 0000 bps Priority: Strict, b/w exceed drops: 0 Priority Level: 2	
Class-map: critical-data (match-all) 45310 packets, 67965000 bytes 5 minute offered rate 719000 bps, drop rate 0000 bps Match: dscp af11 (10)	This is start of section for class named critical-data. As always the classification statistics and criteria are shown first.
Queueing queue limit 2083 packets (queue depth/total drops/no-buffer drops) 0/0/0 (pkts output/bytes output) 45310/67965000 bandwidth 50% (500000 kbps)	Since this class has the bandwidth action a queue is created for the class. This section shows the queue related configuration and statistics.
Class-map: class-default (match-any) 51513 packets, 77222561 bytes 5 minute offered rate 194000 bps, drop rate 0000 bps Match: any	This is the start of the section for class-default, the implicit class that exists in every policy. As always we first show the statistics for packets deemed to belong to this class.
queue limit 4166 packets (queue depth/total drops/no-buffer drops) 0/0/0 (pkts output/bytes output) 1371790/2057638061	This section shows the queue information for class-default.

As mentioned above the **show policy-map interface** command receives an update from the data plane every 10 seconds.

It is also possible to look directly at the data plane for a real time view of system behavior. The command will also allow you to verify the data plane is programmed as expected.

Counters for long term events such as packets enqueued or packets dropped will be cleared each time that information is pushed to the control plane, every 10 seconds.

Perhaps the most useful counter in the data plane show output is the instantaneous queue depth. As this is read each time you issue the command you can get realtime visibility into whether a queue is congested, is it sustained congestion or bursty behavior etc.

The **show platform hardware qfp active feature qos interface** *interface* is the command to view QoS configuration and statistics in the hardware data plane.

The following show example output from the command corresponding to the configuration and **show policy-map interface** example above.

You can see the output of the command again reflects the structure of the policy-map with a section for each class configured.

```
Device#show platform hardware qfp active
feature qos interface gig1/0/4
Interface: GigabitEthernet1/0/4, QFP
interface: 11
  Direction: Output
    Hierarchy level: 0
    Policy name: show_policy-example
```

```
      Class name: voice, Policy name:
show_policy-example
        Police:
            cir:  100096000 bps,  bc: 63488
 bytes
            pir: 0 bps,  be: 0 bytes
            rate mode: Single Rate Mode
            conformed: 0 packets, 0 bytes;
actions:
              transmit
            exceeded: 0 packets, 0 bytes;
actions:
              drop
            violated: 0 packets, 0 bytes;
actions:
              drop
            color aware: No
            green_qos_group: 0,
yellow_qos_group: 0
            overhead accounting: disabled
           overhead value: 0, overhead atm:
 No
```

Start of section for class named voice.

Policer configuration and statistics for current 10 second interval

```
Queue: QID: 175 (0xaf)
            bandwidth (cfg)   : 0
         , bandwidth (hw)    : 0
            shape (cfg)       : 0
         , shape (hw)        : 0
            prio level (cfg)  : 1
         , prio level (hw)   : 0
            limit (pkts )     : 512
          drop policy: tail-drop
          Statistics:
            depth (pkts )     : 0
            tail drops (bytes): 0
         , (packets)         : 0
            total enqs (bytes): 0
         , (packets)         : 0
            licensed throughput
oversubscription drops:
                       (bytes): 0
         , (packets)         : 0
          Schedule: (SID:0x258)
            Schedule FCID     : 16
            bandwidth (cfg)   : 1050 Mbps
         , bandwidth (hw)    : 1050.01 Mbps
            shape (cfg)       : 1050 Mbps
         , shape (hw)        : 1050.01 Mbps
```

Queue information for class voice

This depth is instantaneous queue depth – can be very useful

Start of section for class-default.

Queue information for this class.

Instantaneous depth and statistics for current 10 second interval

```
Class name: class-default, Policy name:
show_policy-example
        Queue: QID: 176 (0xb0)
            bandwidth (cfg)   : 0
         , bandwidth (hw)    : 0
            shape (cfg)       : 0
         , shape (hw)        : 0
            prio level (cfg)  : 0
         , prio level (hw)   : n/a
            limit (pkts )     : 4166
          drop policy: tail-drop
          Statistics:
            depth (pkts )     : 0
            tail drops (bytes): 0
         , (packets)         : 0
            total enqs (bytes): 3420000
         , (packets)         : 2280
            licensed throughput
oversubscription drops:
                       (bytes): 0
         , (packets)         : 0
```

```
Class name: video, Policy name:
show_policy-example
        Police:
            cir:  200064000 bps,  bc: 253952
 bytes
            pir: 0 bps,  be: 0 bytes
            rate mode: Single Rate Mode
            conformed: 0 packets, 0 bytes;
actions:
              transmit
            exceeded: 0 packets, 0 bytes;
actions:
              drop
            violated: 0 packets, 0 bytes;
actions:
              drop
            color aware: No
            green_qos_group: 0,
yellow_qos_group: 0
            overhead accounting: disabled
            overhead value: 0, overhead atm:
 No
```

Start of section for class named video.

Admission control policer configuration and statistics are shown first.

Queue information for class video

Instantaneous queue depth and statistics for current 10 second interval

```
Queue: QID: 178 (0xb2)
            bandwidth (cfg)   : 0
         , bandwidth (hw)    : 0
            shape (cfg)       : 0
         , shape (hw)        : 0
            prio level (cfg)  : 2
         , prio level (hw)   : 1280
            limit (pkts )     : 512
          drop policy: tail-drop
          Statistics:
            depth (pkts )     : 0
            tail drops (bytes): 0
         , (packets)         : 0
            total enqs (bytes): 0
         , (packets)         : 0
            licensed throughput
oversubscription drops:
                       (bytes): 0
         , (packets)         : 0
          Schedule: (SID:0x258)
            Schedule FCID     : 16
            bandwidth (cfg)   : 1050 Mbps
         , bandwidth (hw)    : 1050.01 Mbps
            shape (cfg)       : 1050 Mbps
         , shape (hw)        : 1050.01 Mbps
```

```
Class name: critical-data, Policy name:
show_policy-example
        Queue: QID: 177 (0xb1)
            bandwidth (cfg)   : 500000000
         , bandwidth (hw)    : 500000000
            shape (cfg)       : 0
         , shape (hw)        : 0
            prio level (cfg)  : 0
         , prio level (hw)   : n/a
            limit (pkts )     : 2083
          drop policy: tail-drop
          Statistics:
            depth (pkts )     : 0
            tail drops (bytes): 0
         , (packets)         : 0
            total enqs (bytes): 0
         , (packets)         : 0
            licensed throughput
oversubscription drops:
                       (bytes): 0
         , (packets)         : 0
```

Start of section for class named critical-data.

Instantaneous queue depth and statistics for current 10 second interval.

Command Reference

Account

Account is not an independent command but rather an extension to scheduling commands that allows a user to specify overhead accounting for that command. Account is presented here to avoid replication in each of the scheduling commands.

Syntax description:

To configure a user defined number of bytes to be added to or subtracted from the scheduling length:

[**no**] **shape** | **bandwidth** *rate* **account user-defined** *value* [**atm**]

To specify encapsulation of a downstream device and automatically calculate the overhead accounting adjustment:

[**no**] **shape | bandwidth** *rate* **account dot1q | qing** *encapsulation*

Command Default:

By default Layer 3 Datagram and Layer 2 headers are included in scheduling calculations.

Usage Guidelines:

If the account option is used in one class containing scheduling actions in a policy-map, the account command with same values must be used in all classes containing scheduling actions. Similarly in a hierarchical policy-map the same account options must be configured in each level of the policy.

Bandwidth

The bandwidth command is used to guarantee a minimum service rate to a class.

Syntax description:

To configure in Kbps:

[**no**] **bandwidth** *rate* [**account** *account options*]

To configure as a percentage of visible bandwidth:

[**no**] **bandwidth percent** *value* [**account** *account options*]

Command Default:

By default there is no minimum bandwidth value configured in the schedule entry for a queue. Note that the default excess weight does guarantee some minimum service.

Usage Guidelines:

The **bandwidth** command may be useful if you have an application for which you know the minimum bandwidth requirements.

Bandwidth rates can be configured in 8Kbps increments and the ASR1K has been tested to achieve accuracy within 1% of those rates.

The **bandwidth** command is only supported in leaf schedules (class layer schedules). If you wish to apportion bandwidth in a parent policy you may use the **bandwidth remaining** command.

If you wish to replicate scheduling behavior of an IOS Classic platform (2 parameter scheduler) you may want to replace all **bandwidth percent** *value* commands in your configuration with **bandwidth remaining percent** *value* command.

Bandwidth remaining

The **bandwidth remaining** command is used to apportion excess bandwidth between classes. It may be configured as a simple weight or as a percentage of available bandwidth.

Syntax Description:

To configure as a simple weight:

[**no**] **bandwidth remaining ratio** *value* [**account** *account options*]

To configure as a percentage:

[**no**] **bandwidth remaining percent** *value* [**account** *account options*]

Command Default:

By default every bandwidth schedule entry, whether in leaf schedule or a parent schedule, is configured with an excess weight of 1. This is equivalent to **bandwidth remaining ratio** *1* being configured in that class.

Usage Guidelines:

Configuring bandwidth remaining as a weight supports values of 1 to 1000. This can allow more granular excess sharing than using the percent option.

Configuring **bandwidth remaining percent** *value* yields behavior similar to IOS classic which used a 2 parameter scheduler.

With a shape on parent / queue on child policy (parent has only class default) **bandwidth remaining ratio** *value* should be used to apportion bandwidth between logical interfaces where parent polices are attached

Fair-Queue

The **fair-queue** command is used to configure flow based fair-queuing in a class configured as a bandwidth queue.

Syntax Description:

fair-queue

Command Default:

By default a single fifo queue is configured for each bandwidth class.

Usage Guidelines:

Flow based fair-queuing is used to ensure a single greedy flow can't consume all the bandwidth allocated to a class.

All packets from any given flow are hashed into the same flow queue.

Flow-queuing should not be configured in a policy attached to a tunnel interface. Since all packets have the same outer header all packets are hashed to the same flow queue thus rendering the feature ineffective.

Priority

The **priority** command is used to give low latency and low jitter treatment to a class of traffic.

Syntax Description:

To configure an absolute priority queue (note should be used with explicit policer)

[**no**] **priority**

To configure an absolute priority queue with multi-level priority queuing (note this should be used with an explicit policer)

[**no**] **priority level** *1 | 2*

To configure a priority queue with conditional policer

[**no**] **priority** *rate in kbps* [**burst in bytes**]

or

[**no**] **priority percent** *rate* [**burst in bytes**]

To configure multilevel priority queueing with conditional policer

[**no**] **priority level** *1 | 2rate in kpbs* [**burst in bytes**]

or

[**no**] **priority level** *1* | *2* **percent***rate* [**burst in bytes**]

Command Default:

By default queues are not configured with priority treatment.

Usage Guidelines:

Priority queues should be used with some form of queue admission control (explicit policer or conditional policer) to avoid chance of starving other classes of service.

The policer conforming burst should be configured to an appropriate value for the application in the queue. The following is a configuration example

```
policy-map always_on_policer_burst_example
    class voice
      priority
      police cir 2000000 1250
```

It is not necessary to configure priority in the parent of a hierarchical policy as priority propagation will ensure packets marked as priority by a leaf schedule will receive priority treatment throughout the scheduling hierarchy.

Shape

Use the **shape** command to configure the maximum rate at which a queue may be serviced. Configuring a shaper does not guarantee throughput to a class, it simply puts an upper bound on the rate at which that class may be serviced.

Syntax Description:

[**no**] **shape average** *rate* [**unit**] [**confirming burst**] [**excess burst**] [**account** *options*]

or

[**no**] **shape average percent** *rate* [**confirming burst**] [**excess burst**] [**account** *options*]

Command Default:

By default there is no maximum rate configured in the schedule entry for a bandwidth queue.

Usage Guidelines:

The **shape** command is most commonly used in a parent policy to limit the rate at which traffic is sent to a remote site.

When used in a parent policy the shape rate will be enforced for all traffic - priority and bandwidth.

The **shape** command has options for conforming and excess burst sizes but these values have no effect on XE platforms. Hardware scheduling on the ASR1K platform obviates the need to optimize burst parameters.

The **shape** command enforces a maximum rate at which a class may be serviced but does not in itself guarantee any throughput to that class. The **bandwidth remaining** command may be used along with the shape command to guarantee throughput.