Weighted Fair Queuing

There are two implementations of Weighted Fair Queuing (WFQ) in IOS:

- Platform-independent WFQ
- Distributed Weighted Fair Queuing (DWFQ)

All Cisco platforms covered in this book, except Cisco 12000, support platform-independent WFQ. DWFQ is available only on VIP interfaces on 7500 series routers when dCEF switching is enabled.

Platform-Independent WFQ

IOS has two implementations of platform-independent WFQ:

- Flow-Based WFQ
- Class-Based CBWFQ

Flow-Based WFQ

Flow-Based WFQ is similar to custom queuing in its effect on traffic streams, but rather than defining a static policy to sort the packets into the individual queues, the streams are sorted automatically. Flow-Based WFQ is a legacy congestion management feature in IOS and has been available since IOS version 11.0; we refer to the legacy Flow-Based WFQ in the rest of this chapter as WFQ.

In WFQ, IP packets are classified into flows using

- ToS bits in the IP header
- IP protocol type
- Source IP address
- Source TCP or UDP socket
- Destination IP address
- Destination TCP or UDP socket

By default, WFQ classifies the traffic streams into 256 different flows; you can change this default using the **dynamic-queues**option of the **fair-queue** configuration command. If more flows exist than queues configured, multiple flows are placed in each queue.

NOTE

WFQ is enabled by default on all serial interfaces with bandwidths (clock rates) less than 2 Mbps. On the Cisco 7500, any special queuing (priority queuing [PQ], custom queuing [CQ], or WFQ) can cause each packet to be copied from an MEMD buffer to a DRAM buffer for processing. Copying data between MEMD and DRAM is very expensive in terms of processing cycles. As a result, use of platform-independent WFQ should be limited to low speed interfaces on the Cisco 7500.

As packets are sorted into queues, they are given weights that are used, in turn, to calculate a sequence number. The sequence number determines the order in which packets are dequeued and transmitted.

These weights are calculated using the following formula:

Weight = 4096 / (IP Precedence + 1)

As an example, assume three streams of packets arrive into a WFQ system:

- A1, A2—500 bytes each with precedence 0
- B1, B2—400 bytes each with precedence 0
- C1—40 bytes with precedence 3

When the first packet, A1, arrives, it's classified and enqueued into subqueue A. WFQ computes a sequence number for A1 using the following formula:

Sequence Number = Current Sequence Number + (weight * length)

Where current sequence number is either

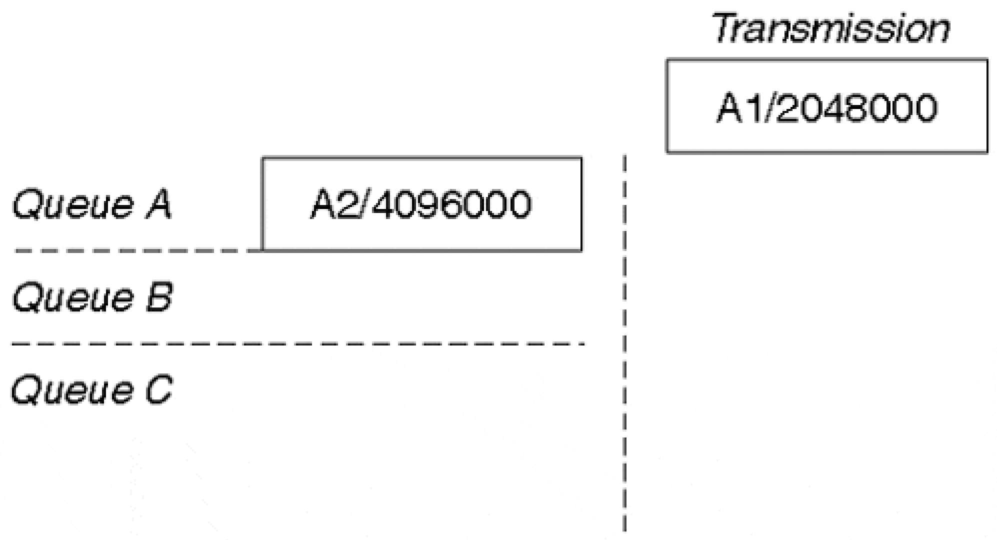
- The same as the sequence number of the packet last enqueued for transmission, if the subqueue is empty.
- The sequence number of the last packet on the subqueue, if the subqueue is not empty.

The current sequence number when A1 arrives is 0 because it's the first packet to arrive into subqueue A. Its weight is 4096 (based on the ToS setting in the packet header) and its length is 500 bytes. Therefore, the sequence number of A1 is calculated as follows:

0 + (4096 * 500) = 2048000

A2 arrives next and WFQ code classifies A2 also into subqueue A. A2 is assigned the sequence number of 4096000 (A1's sequence number + 4096 * 500). A1 is scheduled to be transmitted next. Figure 8-6 shows the queues at this point.

Figure 8-6. WFQ State after the First Pass



B1 and B2 arrive next and WFQ classifies them into subqueue B and computes sequence numbers for them as described in the paragraphs that follow.

Subqueue B is empty when B1 arrives into subqueue B, so B1's sequence number is computed using the sequence number of the last packet enqueued for transmission (A1):

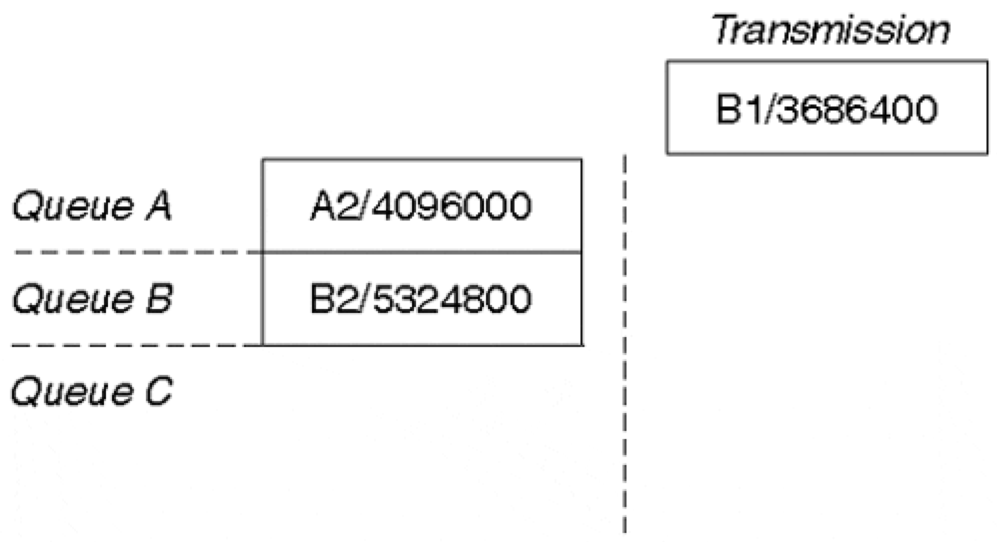
2048000 + (4096 * 400) = 3686400

B2's sequence number is computed similar to A2's:

3686400 + (4096 * 400) = 5324800

At this point, A1 finishes transmission. WFQ sorts the subqueues based on the sequence numbers of the packets at the head of the queues. B1 has the smallest sequence number and is enqueued for transmission. Note that B1 is enqueued for transmission *before* A2, even though it arrived later. Figure 8-7 shows the queues at this point.

Figure 8-7. WFQ State

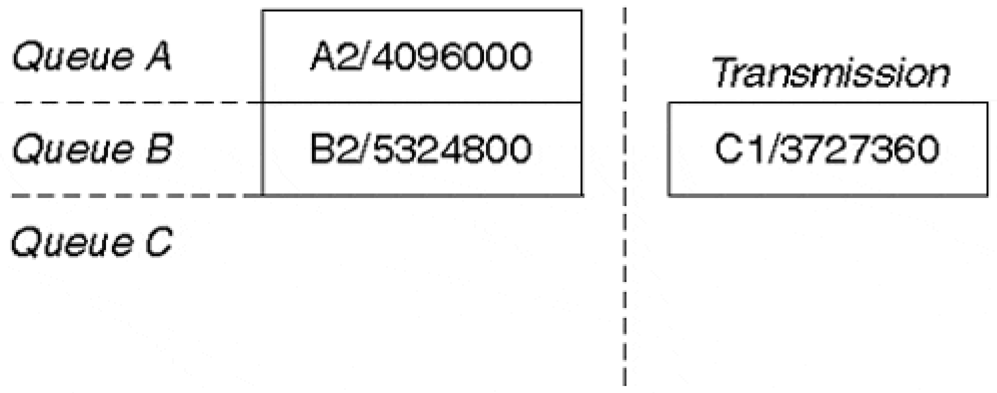


C1 arrives next and is placed into subqueue C. Because subqueue C is empty, C1's sequence number is based on the sequence number of the last packet enqueued for transmission, which was B1.

3686400 + (1024 * 40) = 3727360

B1 is transmitted at this point and all the queues are resorted again. C1 now has the lowest sequence number, as shown in Figure 8-8, so it's enqueued for transmission.

Figure 8-8. WFQ State after the Second Pass



Note that C1 is transmitted *before* A2 or B2, even though it arrived after these two packets.

So what is the purpose behind all this weighting and sorting? The answer is to provide a fair amount of bandwidth for each flow based on its type of service (ToS). Because packet lengths are taken into account when assigning sequence numbers, flows with shorter than average packet lengths can send more packets than flows with longer than average packet lengths. For example, a flow that normally sends 500 byte packets can transmit three times as many packets than a flow that normally sends 1500 byte packets.

The classic example of how this is useful is Telnet traffic and FTP traffic. Telnet traffic uses small packets and the users are affected (and annoyed) by delays. FTP traffic uses large packets and is more tolerant of delays. WFQ automatically gives Telnet packets an earlier transmission time than an FTP packet that arrived slightly before it.

If the flows are weighted based on a type of service, flows with a greater type of service setting receive a greater amount of bandwidth.

Configuring and Monitoring WFQ

To enable WFQ on an interface, configure **fair-queue** at the interface level:

```
Router(config-if)# fair-queue [congestive-discard-threshold [dynamic-queues
[reservable-queues]]]
```

The *congestive-discard-threshold* (the default is 64)is the number of messages or packets allowed in each queue. When this threshold is reached, any packets that normally are placed into the queue are dropped.

dynamic-queues(the default is 256) are the WFQ subqueues into which packets are classified.

If a flow has a reservation through RSVP, its traffic is placed into a *reservable-queue*. The **show queueing** *interface-number* command can monitor the status of WFQ on an interface, as demonstrated in Example 8-7.

Example 8-7. *show queueing* Monitors WFQ Status on a Router Interface

Router#**show queueing serial0** Input queue: 0/75/0 (size/max/drops); Total output drops: 0 Queueing strategy: weighted fair Output queue: 3/1000/64/0 (size/max total/threshold/drops) Conversations 1/3/256 (active/max active/threshold) Reserved Conversations 0/1 (allocated/max allocated) (depth/weight/discards/tail drops/interleaves) 2/4096/0/0/0 Conversation 1023, linktype: ip, length: 1504 source: 10.1.1.1, destination: 10.1.5.2, id: 0xC4FE, ttl:120, ToS: 0 prot: 6, source port 1520, destination port 4563

The following list explains the relevant fields from the output of the **show queueing serial0**. The shaded portion of Example 8-7 highlights the statistics for a single flow or conversation; a conversation queue is the same as a subqueue discussed in the example previously.

- **Output queue: 3/1000/64/0 (size/max total/threshold/drops)**
    - **size** is the instantaneous value of the packets in the queuing system.
    - **max total** is the per interface limit on the total number of buffers that WFQ can consume.
    - **threshold** is the congestive discard threshold described earlier. This is the maximum number of packets allowed in the sub-queue; this is a soft limit—more than **threshold** number of packets actually can be placed in a conversation queue. The **max total**number protects the WFQ system from consuming lots of memory and dropping packets when the max total is reached.
    - **drops** is the number of packets dropped by WFQ either for a sub-queue exceeding the **threshold**or the total number of packets in the queues exceeding the **max total**.
- **Conversations 1/3/256 (active/max active/threshold)**
    - **active** indicates the number of flows or conversations active at this instant. The example shows 1 active conversation.
    - **max active** is the total number of conversations ever.
    - **threshold** is the maximum number of configured conversation or flow queues. If more conversations exist than queues configured, multiple conversations are placed in a queue.
- **(depth/weight/discards/tail drops/interleaves) 2/4096/0/0/0**
    - **depth** is the number of packets placed into this conversation queue.
    - **weight** indicates the weight of each packet in the queue.
    - **discards** is the number of packets dropped due to congestive discard threshold exceeded.
    - **tail drops** is the number of packets dropped when the max-total was exceeded.
    - **interleaves** occur when link layer fragmentation and interleaving is configured. This allows smaller packets to be interleaved between the fragments of larger packets.

Class-Based WFQ

Flow-Based WFQ, as discussed in the previous section, automatically classifies flows—you don't have any control over the classification mechanism. Class-Based WFQ (CBWFQ) was developed to provide a WFQing mechanism that allows flows to be sorted along user-designated policies. CBWFQ, like Flow-Based WFQ, is a platform-independent feature that is available on all router platforms covered in this book except the Cisco 12000. Distributed CBWFQ (DCBWFQ) also is available on VIP cards on 7500 series routers. This section deals only with platform-independent CBWFQ; DCBWFQ is covered in the section, "Distributed Weighted Fair Queuing."

NOTE

As of this writing, CBWFQ is a relatively new feature; please check the IOS Release s for information specific to CBWFQ.

CBWFQ allows you to classify traffic into (up to) 64 classes based on such criteria as the protocol, the access control lists, and the input interface. Each class is allocated a percentage of the bandwidth available on the outbound link. By default, packets that overflow any of the traffic class queues are tail dropped, but Weighted Random Early Detection (WRED), a random drop algorithm discussed later in this chapter, can be configured instead to manage these drops within each queue.

A default class also can be configured; if no default class is configured, packets that are not otherwise classified into a user-defined class are sorted by flow and are given best-effort treatment.

NOTE

CBWFQ allows you to configure the bandwidth allocated per class. If a class called "voice" is allocated 200 kbps of bandwidth but has no traffic, the 200 kbps is distributed among all the other classes in proportion to the bandwidth configured for the other classes.

Modular CLI for CBWFQ Configuration

CBWFQ configuration uses a new modular command-line interface (CLI), which is being standardized across all the IOS QoS features and platforms. The modular CLI provides a clean separation between the specification of a classification policy and the specification of other policies that act based on the results of the applied classification. The modular CLI has three constructs:

- **Class-map—**

  Classifies packets into classes.

- **Policy-map—**

  Describes the policy for each class.

- **Service-policy—**

  Applies the policies defined to an interface.

Example 8-8 provides a sample CBWFQ configuration.

Example 8-8. Sample CBWFQ Configuration

class-map class1 match access-group 101 ! policy-map policy1 class class1 bandwidth 100 queue-limit 20 class class-default bandwidth 200 random-detect ! interface serial0 service-policy output policy1 ! access-list 101 permit ip any any precedence critical

class-map defines a user-defined class called class1. Traffic is classified into the class1 queue by **access-list 101**.

policy-map applies policies to the configured classes. In this example, class1 is allocated 100 kbps of bandwidth (when the interface is congested). Bandwidth also can be allocated as a percentage using the **percent**option in the **bandwidth** command, as shown in Example 8-9.

Example 8-9. Configuring Bandwidth as a Percentage

Router#**configure terminal** Enter configuration commands, one per line. End with CNTL/Z. Router(config)#**policy-map policy1** Router(config-pmap)#**class class1** Router#**bandwidth percent ?** <1-100> Percentage Router#

The **queue-limit** command determines the maximum number of packets that can be queued on the class queue (the default and the maximum is 64). A **policy-map** can apply policy for many classes (up to 64). Note that each **policy-map** command in Example 8-9 is associated with a previously defined class, tying a set of policies to a defined class.

The **class-default** option of the **class**command defines the policy applied to packets that don't fall into any other configured classes. WRED can be configured per class using the **random-detect** command under the class.

After specifying how the packets should be classified, and actions to take on the packets in each class, use the **service-policy** command to apply the policies to the traffic passing through an interface. An additional keyword (**input**/**output**) must be specified to indicate whether the policies apply to packets coming in (or going out of) the interface. CBWFQ can be applied only on packets outbound or an interface.

Monitoring CBWFQ

The **show policy** *policy-map* command can be used to display the configuration of all the classes in a policy map. Example 8-10 shows the output of this command:

Example 8-10. CBWFQ: *show policy* Output

```
ubrl19-2#show policy policy1 Policy Map policy1 class class1 Class class1 Bandwidth 100 (kbps) Max
Thresh 20 (packets) class class-default Class class-default Bandwidth 200 (kbps) exponential weight 9 class
min-threshold max-threshold mark-probability ---------------------------------------------------------- 0 - - 1/10 1 - -
1/10 2 - - 1/10 3 - - 1/10 4 - - 1/10 5 - - 1/10 6 - - 1/10 7 - - 1/10 rsvp - - 1/10
```

The random early detection (RED) parameters under **class-default** are explained in the section "Weighted Random Early Detection," later in this chapter.

The output of **show policy** *interface* is the key to determining how CBWFQ is functioning on the interface. Example 8-11 displays the output of this command:

Example 8-11. Output of *show policy interface* to Monitor CBWFQ

```
ubrl19-2#show policy int Serial0 service-policy output: policy1 class-map: class1 (match-all) 0 packets, 0
bytes 5 minute offered rate 0 bps, drop rate 0 bps match: access-group 101 Output Queue: Conversation
265 Bandwidth 100 (kbps) Packets Matched 0 Max Threshold 20 (packets) (discards/tail drops) 0/0 class-
map: class-default (match-any) 0 packets, 0 bytes 5 minute offered rate 0 bps, drop rate 0 bps match: any 0
packets, 0 bytes 5 minute rate 0 bps Output Queue: Conversation 266 Bandwidth 200 (kbps) Packets
Matched 326 random-detect: mean queue depth: 0 drops: class random tail min-th max-th mark-prob 0 0 0
20 40 1/10 1 0 0 22 40 1/10
```

The following list explains the relevant output from Example 8-11:

- **Bandwidth 100 (kbps) Packets Matched 0 Max Threshold 20 (packets)**
  - **Packets Matched** is the number of packets that matched this class policy.
  - **Max Threshold** is the same as the congestive discard threshold in the WFQ section; it's a soft limit on the number of packets that can be placed in the class queue.
- **(discards/tail drops) 0/0**
  - **discards** is the number of packets dropped when Max Threshold is reached.
  - **tail drops** is the number of packets dropped when max-total is reached; the default of max-total is 1000 and cannot be changed currently.
  - The RED parameters are explained in the section "Weighted Random Early Detection," later in

the chapter.

Low Latency Queuing

The Low Latency Queuing (LLQ) feature combines strict priority queuing with Class-Based Weighted Fair Queuing (CBWFQ). Strict priority queuing allows delay-sensitive data, such as voice traffic, to be dequeued and sent first (before packets in other queues are dequeued). This feature can reduce jitter for voice traffic. To enqueue a class of traffic to the strict priority queue, you configure the **priority** command for the class after you specify the named class within a policy map (classes to which the priority command is applied are considered priority classes).

Within a policy map, you can give one or more classes priority status. When multiple classes within a single policy map are configured as priority classes, all the traffic from these classes is enqueued to the same, single, strict priority queue. Example 8-12 shows class1 from Example 8-11 now configured as a priority queue.

Example 8-12. Configuring a Priority Queue in CBWFQ for LLQ

```
policy-map policy1 class class1 priority 100 class class-default bandwidth 200 random-detect
```

Distributed Weighted Fair Queuing

Distributed Weighted Fair Queuing (DWFQ) is implemented on the VIP on the Cisco 7500. Platform-independent queuing mechanisms (PQ, CQ, or WFQ) do not scale to high-speed interfaces (greater than 2 Mbps) on the 7500 platform. The main reason for this performance limitation is that packets copied back and forth between MEMD and DRAM becomes a CPU-intensive operation when interface congestion occurs and any fancy queuing mechanism is configured.

DWFQ is designed for distributed switching on VIPs and greatly improves WFQ performance.

- All packet processing is performed on the VIP's processor, which frees the RSP's processor up for other tasks (such as running routing protocols).
- Packets aren't copied to and from various types of memory within the router; all queuing takes place within SRAM on the VIPs themselves.
- The algorithm used for DWFQ has been optimized for the VIP processor's environment and interface speeds.

As a result, DWFQ can be enabled on such high speed interfaces as the OC-3 (155 Mbps) interface on a VIP2-50. DWFQ is capable of classifying packets in four different ways:

- Flow-Based
- ToS-Based
- QoS Group–Based
- CBWFQ

Flow-Based DWFQ

Flow-Based DWFQ classifies the traffic destined out an interface based on the following:

- ToS bits in the IP header
- IP protocol type

- Source IP address
- Source TCP or UDP socket
- Destination IP address
- Destination TCP or UDP socket

Traffic is classified into 512 flows; each flow is serviced round-robin from the calendar queue (see the "DWFQ Implementation" section later in the chapter for more details). Each of the 512 flows shares an equal amount of bandwidth.

With distributed Cisco Express Forwarding (dCEF) enabled on the interface, configuring **fair-queue** on a VIP interface enables Flow-Based DWFQ. To configure DWFQ on an interface, use the command **fair-queue** on an interface for which dCEF is configured already:

```
Router(config-if)#fair-queue
```

ToS-Based DWFQ

ToS-Based DWFQ, also known as Class-Based DWFQ, classifies packets based on the lower two bits of the precedence field in the IP header, so there are four possible queues. The weight of each queue determines the amount of bandwidth the traffic placed in this queue receives if the outbound link is fully congested.

The following command enables ToS DWFQ:

```
Router(config-if)#fair-queue tos
```

Class 3, 2, 1, and 0 get default weights of 40, 30, 20, and 10, respectively. The default weights can be changed using

```
Router(config-if)#fair-queue tos 0 weight 20
```

The aggregate weight of all classes cannot exceed 100.

QoS Group–Based DWFQ

Another variant of ToS-Based (or Class-Based) DWFQ is based on QoS groups instead of the ToS bits in the IP header. QoS groups are created by local policies applied through Committed Access Rate (CAR) and can be propagated through Border Gateway Protocol (BGP) Policy Propagation. These concepts are beyond the scope of this book.

Distributed CBWFQ

Distributed ToS-based and QoS group–based WFQ are two forms of *CBWFQ*. The platform independent form of CBWFQ using the same modular CLI can also be configured on a VIP interface, including the Low Latency queuing feature.

DWFQ Implementation

DWFQ does not perform list sorting and re-ordering of packets like the platform-independent WFQ algorithm does. Instead, the DWFQ scheduling algorithm uses calendar queues to achieve similar behavior with much greater efficiency. This is the primary reason DWFQ can be enabled on an OC-3 (155-Mbps) interface with a VIP2-50.

The difference between the platform-independent WFQ and DWFQ is in the details of how the sorted queue is managed (that is, the difference is in the implementation details, not in the fundamentals of the algorithm). WFQ maintains a sorted, linked list where newly arriving packets are inserted into the sorted list based on the timestamp assigned to the packet. In DWFQ, a calendar queue is used to impose the sorted ordering. The calendar queue is more efficient in terms of CPU utilization.

Monitoring DWFQ

The primary command used to monitor the state of DWFQ is as demonstrated in Example 8-13.

Example 8-13. *show interface fair* Monitors DWFQ State

```
router#show interface fair FastEthernet11/1/0 queue size 0 packets output 44404, wfq drops 1, nobuffer
drops 0 WFQ: aggregate queue limit 6746, individual queue limit 3373 max available buffers 6746 Class 0:
weight 10 limit 3373 qsize 0 packets output 44404 drops 1 Class 1: weight 20 limit 3373 qsize 0 packets
output 0 drops 0 Class 2: weight 30 limit 3373 qsize 0 packets output 0 drops 0 Class 3: weight 40 limit 3373
qsize 0 packets output 0 drops 0
```

The following list explains the relevant fields in the output of the **show interface fair**:

- **Packets output—**

  Total number of packets transmitted out this interface.

- **wfq drops—**

  Packets dropped by DWFQ; an aggregate of the drops in each class. The drops could be due to three reasons:

  - The aggregate queue limit was exceeded.
  - The individual queue limit was exceeded (this is not enforced until two-thirds of the aggregate queue limit is reached).
  - There were **no buffers** available.

- **nobuffer drops—**

  Indicates the VIP is running out of SRAM and cannot enqueue packets to the DWFQ system. The total numbers of buffers available to DWFQ is indicated by **max available buffers**.

- **aggregate queue limit—**

  The maximum number of packets that can be enqueued to the DWFQ system. By default, this is equal to the but it can be tuned.

- **max available buffers—**

  This is the amount of SRAM carved for use by DWFQ. It is computed as a function of the bandwidth of the interface and the maximum available SRAM.

  - If there is sufficient SRAM available, then the aggregate queue limit assigned to an interface is equal to the number of buffers needed to hold half a second worth of traffic, assuming each packet is 250 bytes.

  - If there isn't enough memory to create a half a second worth of buffers, memory is allocated based on a complex algorithm.

- **individual queue limit—**

  The number of packets that are allowed in each class queue; half the aggregate limit by default. This is a soft limit, and is not enforced until two-thirds of the aggregate queue limit is reached. By default, the sum of the individual queue limits of each class exceeds the aggregate queue limit; this is intentional oversubscription and can be tuned.

  NOTE

  If many large packets belonging to a particular class (class1) arrive in a burst followed by a few packets of a second class (class2), it is possible for all the buffers to be consumed by the class1 packets, causing the class2 packets to be dropped due to **no buffers**. Possible workarounds are to add SRAM on the VIP or to tune down the aggregate limit and the individual limits so buffers are not exhausted. If you want to fully protect every class, then the sum of the individual limits must be less than the aggregate limit. In other words, you can't allow any oversubscription of the buffers if you want to guarantee one class won't starve another class.

- **qsize—**

  Instantaneous depth of the class queue.

< BACK Make Note | Bookmark CONTINUE >

Index terms contained in this section