- QoS and Queuing -

Queuing Overview

A **queue** is used to store traffic until it can be processed or serialized. Both switch and router interfaces have **ingress** (*inbound*) **queues** and **egress** (*outbound*) **queues**.

An ingress queue stores packets until the switch or router CPU can forward the data to the appropriate interface. An egress queue stores packets until the switch or router can *serialize* the data onto the physical wire.

Switch ports and router interfaces contain both **hardware** and **software** queues. Both will be explained in detail later in this guide.

Queue Congestion

Switch (and router) queues are susceptible to **congestion**. Congestion occurs when the rate of *ingress* traffic is greater than can be successfully processed and serialized on an egress interface. Common causes for congestion include:

- The speed of an *ingress* interface is higher than the *egress* interface.
- The combined traffic of multiple ingress interfaces exceeds the capacity of a single egress interface.
- The switch/router CPU is insufficient to handle the size of the forwarding table.

By default, if an interface's queue buffer fills to capacity, new packets will be dropped. This condition is referred to as **tail drop**, and operates on a first-come, first-served basis. If a standard queue fills to capacity, any new packets are indiscriminately dropped, regardless of the packet's classification or marking.

QoS provides switches and routers with a mechanism to **queue** and **service** *higher* priority traffic before *lower* priority traffic. This guide covers various queuing methods in detail.

QoS also provides a mechanism to **drop** *lower* priority traffic before *higher* priority traffic, during periods of congestion. This is known as Weighted Random Early Detection (WRED), and is covered in detail in another guide.

* * *

Types of Queues

Recall that interfaces have both **ingress** (*inbound*) queues and **egress** (*outbound*) queues. Each interface has one or more **hardware queues** (also known as **transmit (TxQ) queues).** Traffic is placed into egress hardware queues to be serialized onto the wire.

There are two *types* of hardware queues. By default, traffic is placed in a **standard queue**, where all traffic is regarded equally. However, interfaces can also support **strict priority** queues, dedicated for higher-priority traffic.

DiffServ QoS can dictate that traffic with a higher DSCP or IP Precedence value be placed in strict priority queues, to be serviced first. Traffic in a strict priority queue is *never dropped* due to congestion.

A Catalyst switch interface may support multiple standard or strict priority queues, depending on the switch model. Cisco notates strict priority queues with a **"p"**, standard queues with a **"q"**, and WRED thresholds per queue (explained in a separate guide) with a **"t".**

If a switch interface supports one strict priority queue, two standard queues, and two WRED thresholds, Cisco would notate this as:

1p2q2t

To view the supported number of hardware queues on a given Catalyst switch interface:

Switch# *show interface fa0/12 capabilities*

The strict priority egress queue must be explicitly *enabled* on an interface:

Switch(config)# *interface fa0/12*
Switch(config-if)# *priority-queue out*

To view the size of the hardware queue of a router serial interface:

Router# *show controller serial*

The size of the interface hardware queue can modified on some Cisco models, using the following command:

Router(config)# *interface serial 0/0*
Router(config-if)# *tx-ring-limit 3*

(Reference: http://www.cisco.com/en/US/tech/tk389/tk813/technologies_tech_note09186a00801558cb.shtml)

* * *

Forms of Queuing

The default form of queuing on nearly all interfaces is **First-In First-Out (FIFO).** This form of queuing requires no configuration, and simply processes and forwards packets in the order that they arrive. If the queue becomes saturated, new packets will be dropped (**tail drop).**

This form of queuing may be insufficient for real-time applications, especially during times of congestion. FIFO will *never* discriminate or give preference to higher-priority packets. Thus, applications such as VoIP can be starved out during periods of congestion.

Hardware queues *always* process packets using the **FIFO** method of queuing. In order to provide a preferred level of service for high-priority traffic, some form of **software queuing** must be used. Software queuing techniques can include:

- First-In First-Out (FIFO) (*default*)
- Priority Queuing (PQ)
- Custom Queuing (CQ)
- Weighted Fair Queuing (WFQ)
- Class-Based Weighted Fair Queuing (CBWFQ)
- Low-Latency Queuing (LLQ)

Each of the above software queuing techniques will be covered separately in this guide.

Software queuing usually employs multiple queues, and each is assigned a specific *priority*. Traffic can then be assigned to these queues, using access-lists or based on classification. Traffic from a higher-priority queue is *serviced* before the traffic from a lower-priority queue.

Please note: traffic *within* a single software queue (sometimes referred to as *sub-queuing*) is always processed using FIFO.

Note also: if the hardware queue is *not congested*, software queues are **ignored.** Remember, software-based queuing is *only* used when the hardware queue is congested. Software queues serve as an *intermediary*, deciding which traffic types should be placed in the hardware queue *first* and *how often*, during periods of congestion.

* * *

Priority Queuing (PQ)

Priority Queuing (PQ) employs four separate queues:

- High
- Medium
- Normal (*default*)
- Low

Traffic must be assigned to these queues, usually using access-lists. Packets from the High queue are *always* processed before packets from the Medium queue. Likewise, packets from the Medium queue are *always* processed before packets in the Normal queue, etc. Remember that traffic *within* a queue is processed using FIFO.

As long as there are packets in the High queue, *no packets* from any other queues are processed. Once the High queue is empty, then packets in the Medium queue are processed… but *only* if no new packets arrive in the High queue. This is referred to as a **strict** form of queuing.

The obvious advantage of PQ is that higher-priority traffic is *always* processed first. The nasty disadvantage to PQ is that the lower-priority queues can often *receive no service at all*. A constant stream of High-priority traffic can starve out the lower-priority queues.

To configure PQ, traffic can first be identified using access-lists:

Router(config)# *access-list 2 permit 150.1.1.0 0.0.0.255*
Router(config)# *access-list 100 permit tcp any 10.1.1.0 0.0.0.255 eq www*

Then, the traffic should be placed in the appropriate queues:

Router(config)# *priority-list 1 protocol ip high list 2*
Router(config)# *priority-list 1 protocol ip medium list 100*
Router(config)# *priority-list 1 protocol ip normal*
Router(config)# *priority-list 1 protocol ipx low*
Router(config)# *priority-list 1 default normal*

The size of each queue (measured in *packets*) can be specified:

Router(config)# *priority-list 1 queue-limit 30 40 50 60*

Finally, the *priority-list* must be applied to an interface:

Router(config)# *interface serial0*
Router(config-if)# *priority-group 1*

* * *

Custom Queuing (CQ)

A less *strict* form of queuing is **Custom Queuing (CQ),** which employs a **weighed round-robin** queuing methodology.

Each queue is processed *in order*, but each queue can have a different *weight* or *size* (measured either in bytes, or the number of packets). Each *queue* processes its entire contents during its *turn.* CQ supports a maximum of **16 queues.**

To configure CQ, traffic must first be identified by *protocol* or with an *access-list,* and then placed in a custom queue:

Router(config)# *access-list 101 permit tcp 172.16.0.0 0.0.255.255 any eq 1982*

Router(config)# *queue-list 1 protocol ip 1 list 101*
Router(config)# *queue-list 1 protocol ip 1 tcp smtp*
Router(config)# *queue-list 1 protocol ip 2 tcp domain*
Router(config)# *queue-list 1 protocol ip 2 udp domain*
Router(config)# *queue-list 1 protocol ip 3 tcp www*
Router(config)# *queue-list 1 protocol cdp 4*
Router(config)# *queue-list 1 protocol ip 5 lt 1000*
Router(config)# *queue-list 1 protocol ip 5 gt 800*

Each custom queue is identified with a number (*1*, *2*, *3* etc.). Once traffic has been assigned to custom queues, then each queue's *parameters* must be specified. Parameters can include:

- A **limit** – size of the queue, measured in number of packets.
- A **byte-count** – size of the queue, measured in number of bytes.

Configuration of queue parameters is straight-forward:

Router(config)# *queue-list 1 queue 1 limit 15*
Router(config)# *queue-list 1 queue 2 byte-count 2000*
Router(config)# *queue-list 1 queue 3 limit 25*
Router(config)# *queue-list 1 queue 4 byte-count 1024*
Router(config)# *queue-list 1 queue 4 limit 10*

Finally, the custom queue must be applied to an interface:

Router(config)# *interface serial0/0*
Router(config-if)# *custom-queue-list 1*

(Reference: http://www.cisco.com/en/US/docs/ios/12_0/qos/configuration/guide/qccq.html)

* * *

Weighted Fair Queuing (WFQ)

Weighted Fair Queuing (WFQ) *dynamically* creates queues based on **traffic flows**. Traffic flows are identified with a hash value generated from the following header fields:

- Source and Destination IP address
- Source and Destination TCP (or UDP) port
- IP Protocol number
- Type of Service value (IP Precedence or DSCP)

Traffics of the same *flow* are placed in the same *flow queue.* By default, a maximum of **256** queues can exist, though this can be increased to **4096**.

If the priority (based on the ToS field) of all packets are the same, bandwidth is divided equally among all queues. This results in low-traffic flows incurring a minimal amount of delay, while high-traffic flows may experience latency.

Packets with a higher priority are *scheduled* before lower-priority packets arriving at the same time. This is accomplished by assigning a **sequence number** to each arriving packet, which is calculated from the last sequence number multiplied by an *inverse weight* (based on the ToS field). In other words a *higher* ToS value results in a *lower* sequence number, and the higher-priority packet will be serviced first.

WFQ is actually the **default** on **slow serial links** (2.048 Mbps or slower). To explicitly enable WFQ on an interface:

Router(config)# *interface s0/0*
Router(config-if)# *fair-queue*

The following are optional WFQ parameters:

Router(config)# *interface s0/0*
Router(config-if)# *fair-queue 128 1024*

The *128* value increases the maximum size of a queue, measured in packets (**64** is the default). The *1024* value increases the maximum number of queues from its default of **256.**

The following queuing methods are based on WFQ:

- Class-Based Weighted Fair Queuing (CBWFQ)
- Low Latency Queuing (LLQ)

* * *

Class-Based WFQ (CBWFQ)

WFQ suffers from several key disadvantages:

- Traffic cannot be queued based on *user-defined* classes.
- WFQ cannot provide specific bandwidth guarantees to a traffic flow.
- WFQ is only supported on slower links (2.048 Mbps or less).

These limitations were corrected with **Class-Based WFQ (CBWFQ).** CBWFQ provides up to **64** user-defined queues. Traffic within each queue is processed using FIFO. Each queue is provided with a configurable minimum bandwidth guarantee, which can be represented one of three ways:

- As a fixed amount (using the *bandwidth* command).
- As a percentage of the total interface bandwidth (using the *bandwidth percent* command).
- As a percentage of the remaining unallocated bandwidth (using the *bandwidth remaining percent* command).

Note: the above three commands must be used exclusively from each other – it is no possible to use the fixed *bandwidth* command on one class, and *bandwidth percent* command on another class within the same policy.

CBWFQ queues are only held to their minimum bandwidth guarantee during periods of congestion, and can thus exceed this minimum when the bandwidth is available.

By default, only 75% of an interface's total bandwidth can be reserved. This can be changed using the following command:

Router(config)# *interface s0/0*
Router(config-if)# *max-reserved-bandwidth 90*

The key disadvantage with CBWFQ is that no mechanism exists to provide a *strict-priority* queue for real-time traffic, such as VoIP, to alleviate latency. Low Latency Queuing (LLQ) addresses this disadvantage, and will be discussed in detail shortly.

* * *

Configuring CBWFQ

CBWFQ is implemented using the Modular Command-Line (MQC) interface. Specifically, a *class-map* is used to identify the traffic, a *policy-map* is used to enforce each queue's bandwidth, and a *service-policy* is used to apply the policy-map to an interface.

```
Router(config)# access-list 101 permit tcp 10.1.5.0 0.0.0.255 any eq http
Router(config)# access-list 102 permit tcp 10.1.5.0 0.0.0.255 any eq ftp

Router(config)# class-map HTTP
Router(config-cmap)# match access-group 101
Router(config)# class-map FTP
Router(config-cmap)# match access-group 102

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth 256
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth 128

Router(config)# interface serial0/0
Router(config-if)# service-policy output THEPOLICY
```

The above example utilizes the *bandwidth* command to assign a fixed minimum bandwidth guarantee for each class. Alternatively, a percentage of the interface bandwidth (75% of the total bandwidth, by default) can be guaranteed using the *bandwidth percent* command:

```
Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth percent 40
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth percent 20
```

The minimum guarantee can also be based as a percentage of the remaining unallocated bandwidth, using the *bandwidth remaining percent* command.

```
Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth remaining percent 20
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth remaining  percent 20
```

Remember, the *bandwidth, bandwidth percent*, and *bandwidth remaining percent* commands must be used exclusively, not in tandem, with each other.

* * *

Low Latency Queuing (LLQ)

Low-Latency Queuing (LLQ) is an improved version of CBWFQ that includes one or more **strict-priority** queues, to alleviate latency issues for real-time applications. Strict-priority queues are *always* serviced before standard class-based queues.

The key difference between LLQ and PQ (which also has a strict priority queue), is that the LLQ strict-priority queue will *not* starve all other queues. The LLQ strict-priority queue is *policed*, either by *bandwidth* or a *percentage* of the bandwidth.

As with CBWFQ, configuration of LLQ is accomplished using MQC:

```
Router(config)# access-list 101 permit tcp 10.1.5.0 0.0.0.255 any eq http
Router(config)# access-list 102 permit tcp 10.1.5.0 0.0.0.255 any eq ftp
Router(config)# access-list 103 permit tcp 10.1.5.0 0.0.0.255 any eq 666

Router(config)# class-map HTTP
Router(config-cmap)# match access-group 101
Router(config)# class-map FTP
Router(config-cmap)# match access-group 102
Router(config)# class-map SECRETAPP
Router(config-cmap)# match access-group 103

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HTTP
Router(config-pmap-c)# bandwidth percent 20
Router(config-pmap)# class FTP
Router(config-pmap-c)# bandwidth percent 20
Router(config-pmap)# class SECRETAPP
Router(config-pmap-c)# priority percent 50

Router(config)# int serial0/1
Router(config-if)# service-policy output THEPOLICY
```

Note that the *SECRETAPP* has been assigned to a strict-priority queue, using the *priority percent* command.

(Reference: http://www.cisco.com/en/US/docs/ios/12_0t/12_0t7/feature/guide/pqcbwfq.html)

* * *

Troubleshooting Queuing

To view the configured queuing mechanism and traffic statistics on an interface:

Router# *show interface serial 0/0*

```
Serial 0/0 is up, line protocol is up
  Hardware is MCI Serial
  Internet address is 192.168.150.1, subnet mask is 255.255.255.0
  MTU 1500 bytes, BW 1544Kbit, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation HDLC, loopback not set
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, output 00:00:01, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: Class-based queueing
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations 0/1/256 (active/max active/max total)
     Reserved Conversations 1/1 (allocated/max allocated)
```

To view the packets currently stored in a queue:

Router# *show queue s0/0*

To view policy-map statistics on an interface:

Router# *show policy-map interface s0/0*

```
Serial0/0

   Service-policy input: THEPOLICY

     Class-map: SECRETAPP (match-all)
       123 packets, 44125 bytes
       1 minute offered rate 1544000 bps, drop rate 0 bps
       Match: access-group 103
       Weighted Fair Queuing
         Strict Priority
         Output Queue: Conversation 264
            Bandwidth 772 (Kbps)
            (pkts matched/bytes matched) 123/44125
```

* * *