# - QoS and Congestion Avoidance -

## ***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. Queuing is covered in detail in a separate guide.

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 this guide.

* * *

## ***Random Early Detection (RED) and Weighted RED (WRED)***

**Tail drop** proved to be an inefficient method of congestion control. A more robust method was developed called **Random Early Detection (RED).** RED prevents the queue from filling to capacity, by randomly dropping packets in the queue. RED essentially takes advantage of TCP's ability to resend dropped packets.

RED helps alleviate two TCP issues caused by tail drop:

- **TCP Global Synchronization –** occurs when a large number of TCP packets are dropped simultaneously. Hosts will reduce TCP traffic (referred to as *slow start)* in response, and then ramp up again… *simultaneously*. This results in cyclical periods of extreme congestion, followed by periods of under-utilization of the link.
- **TCP Starvation** – occurs when TCP flows are *stalled* during times of congestion (as detailed above), allowing non-TCP traffic to saturate a queue (and thus starving out the TCP traffic).

RED will randomly drop queued packets based on configurable *thresholds.* By dropping only *some* of the traffic before the queue is saturated, instead of *all* newly-arriving traffic (*tail drop)*, RED limits the impact of TCP global synchronization.

RED will drop packets using one of three methods:

- **No drop** – used when there is no congestion.
- **Random drop** – used to prevent a queue from becoming saturated, based on thresholds.
- **Tail drop** – used when a queue *does* become saturated.

RED *indiscriminately* drops random packets. It has no mechanism to differentiate between traffic flows. Thus, RED is mostly deprecated.

**Weighted Random Early Detection (WRED)** provides more granular control – packets with a lower IP Precedence or DCSP value can be dropped *more frequently* than higher priority packets.

This guide will concentrate on the functionality and configuration of WRED.

* * *

## ***WRED Fundamentals***

There are two methods to configuring WRED. **Basic WRED configuration** is accomplished by configuring minimum and maximum packet *thresholds* for each IP Precedence or DSCP value.

- The **minimum threshold** indicates the minimum number of packets that must be queued, before packets of a specific IP Precedence or DSCP value will be *randomly* dropped.
- The **maximum threshold** indicates the number of packets that must be queued, before *all* new packets of a specific IP Precedence or DSCP value are dropped. When the maximum threshold is reached, WRED essentially mimics the tail drop method of congestion control.
- The **mark probability denominator (MPD)** determines the number of packets that will be dropped, when the size of the queue is in between the minimum and maximum thresholds. This is measured as a fraction, specifically 1/MPD. For example, if the MPD is set to 5, one out of every 5 packets will be dropped. In other words, the chance of each packet being dropped is 20%.

Observe the following table:

| Precedence | Minimum Threshold | Maximum Threshold | MPD |
|---|---|---|---|
| 0 | 10 | 25 | 5 |
| 1 | 12 | 25 | 5 |
| 2 | 14 | 25 | 5 |
| 3 | 16 | 25 | 5 |

If the WRED configuration matched the above, packets with a precedence of *0* would be randomly dropped once *10* packets were queued. Packets with a precedence of *2* would similarly be dropped once *14* packets were queued. The maximum queue size is *25*, thus all new packets of any precedence would be dropped once *25* packets were queued.

**Advanced WRED configuration** involves tuning WRED maximum and minimum thresholds on a per-queue basis, rather than to specific IP Precedence or DSCP values. In this instance, the min and max thresholds are based on *percentages*, instead of a specific number of packets. This is only supported on higher model Catalyst switches.

WRED only affects standard queues. Traffic from strict priority queues is never dropped by WRED.

* * *

## ***Configuring Basic WRED***

WRED configuration can be based on either **IP Precedence** or a **DSCP value**. To configure WRED thresholds using IP Precedence:

```
Router(config)# interface fa0/1
Router(config-if)# random-detect
Router(config-if)# random-detect precedence 0 10 25 5
Router(config-if)# random-detect precedence 1 12 25 5
Router(config-if)# random-detect precedence 2 14 25 5
Router(config-if)# random-detect precedence 3 16 25 5
Router(config-if)# random-detect precedence 4 18 25 5
Router(config-if)# random-detect precedence 5 20 25 5
```

The first *random-detect* command enables WRED on the interface. The subsequent *random-detect* commands apply a minimum threshold, maximum threshold, and MPD value, for each specified IP Precedence level.

To configure WRED thresholds using DSCP values:

```
Router(config)# interface fa0/10
Router(config-if)# random-detect
Router(config-if)# random-detect dscp-based af11 14 25 5
Router(config-if)# random-detect dscp-based af12 12 25 5
Router(config-if)# random-detect dscp-based af13 10 25 5
Router(config-if)# random-detect dscp-based af21 20 25 5
Router(config-if)# random-detect dscp-based af22 18 25 5
Router(config-if)# random-detect dscp-based af23 16 25 5
```

To view the WRED status and configuration on all interfaces:

```
Router# show interface random-detect
Router# show queuing
```

WRED is **not compatible** with Custom Queuing (CQ), Priority Queuing (PQ) or Weighted Fair Queuing (WFQ), and thus *cannot be enabled* on interfaces using one of those queuing methods.

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

* * *

## ***Configuring Advanced WRED with WRR***

On higher-end Catalyst models, WRED can be handled on a per-queue basis, and is configured in conjunction with a feature called **Weighted Round Robin (WRR)**.

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*

* * *

## ***Configuring Advanced WRED with WRR(continued)***

Standard egress queues can be assigned **weights**, which dictate the proportion of traffic sent across each queue:

**Switch(config-if)#** *wrr-queue bandwidth 127 255*

The above command would be used if a particular port has two standard egress queues (remember, the number of queues depends on the Catalyst model). The two numbers are the weights for Queue 1 and Queue 2, respectively. The weight is a number between 1 and 255, and serves as a ratio for sending traffic.

In the above example, Queue 2 would be allowed to transmit twice as much traffic as Queue 1 every **cycle** (*255* is roughly twice that of *127*). This way, the higher-priority traffic should always be serviced first, and more often.

Next, WRED/WRR can be enabled for a particular queue. Cisco's documentation on this is inconsistent on whether it is enabled by default, or not. To manually enable WRED/WRR on Queue 1:

**Switch(config-if)#** *wrr-queue random-detect 1*

To disable WRED/WRR and revert to tail-drop congestion control:

**Switch(config-if)#** *no wrr-queue random-detect 1*

Next, the WRED/WRR minimum and maximum thresholds must be tuned. Again, this is accomplished per standard queue, and based on a *percentage* of the capacity of the queue.

Recall that each switch port has a specific set of queues (for example, **1p2q2t**). The **2t** indicates that two WRED/WRR thresholds can exist per standard queue.

**Switch(config-if)#** *wrr-queue random-detect min-threshold 1 5 10*
**Switch(config-if)#** *wrr-queue random-detect max-threshold 1 40 100*

The first command sets two separate *min-threshold*s for Queue *1*, specifically *5* percent and *10* percent.

The second command sets two separate *max-threshold*s for Queue *1*, specifically *40* percent and *100* percent.

* * *

## ***Configuring Advanced WRED with WRR (continued)***

Why two separate minimum and maximum thresholds per queue? Because packets of a specific CoS value can be mapped to a specific threshold of a specific queue.

Observe:

**Switch(config-if)#** *wrr-queue cos-map 1 1 0 1*
**Switch(config-if)#** *wrr-queue cos-map 1 2 2 3*

The first command creates a *map*, associating queue *1*, threshold *1* with CoS values of *0* and *1*.

The second command creates a *map*, associating queue *1,* threshold *2* with CoS values of *2* and *3*.

All traffic marked with CoS value 0 or 1 will have a minimum threshold of 5 percent, and a maximum threshold of 40 percent (per the earlier commands). All traffic marked with CoS value 2 or 3 will have a minimum threshold of 10 percent, and a maximum threshold of 100 percent.

The above *wrr-queue* commands are actually the *default* settings on higher-end Catalyst switches.

To view the QoS settings on a Catalyst interface:

**Switch#** *show mls qos interface fa0/10*

To view the queuing information for a Catalyst interface:

**Switch#** *show mls qos interface fa0/10 queuing*

To view QoS mapping configurations:

**Switch#** *show mls qos maps*

* * *

## ***Configuring Class-Based WRED (CBWRED)***

The functionality of Class-Based Weighted Fair Queuing (CBWFQ) can be combined with WRED to form **Class-Based WRED (CBWRED).** CBWFQ is covered in detail in a separate guide.

CBWRED is implemented within a policy-map:

```
Router(config)# class-map HIGH
Router(config-cmap)# match ip precedence 5
Router(config)# class-map LOW
Router(config-cmap)# match ip precedence 0 1 2

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HIGH
Router(config-pmap-c)# bandwidth percent 40
Router(config-pmap-c)# random-detect
Router(config-pmap-c)# random-detect precedence 5 30 50 5

Router(config-pmap)# class LOW
Router(config-pmap-c)# bandwidth percent 20
Router(config-pmap-c)# random-detect
Router(config-pmap-c)# random-detect precedence 0 20 50 5
Router(config-pmap-c)# random-detect precedence 1 22 50 5
Router(config-pmap-c)# random-detect precedence 2 24 50 5

Router(config)# int fa0/1
Router(config-if)# service-policy output THEPOLICY
```

DSCP values can be used in place of IP Precedence:

```
Router(config)# class-map HIGH
Router(config-cmap)# match ip dscp af31 af41

Router(config)# policy-map THEPOLICY
Router(config-pmap)# class HIGH
Router(config-pmap-c)# bandwidth percent 40
Router(config-pmap-c)# random-detect dscp-based
Router(config-pmap-c)# random-detect dscp af31 28 50 5
Router(config-pmap-c)# random-detect dscp af41 30 50 5
```

To view CBWRED statistics:

```
Router# show policy-map
```

* * *