Chris Bryant, CCIE #12933 **http://www.thebryantadvantage.com**

Back To Index

Congestion Detection And Avoidance

Overview

The Danger of Tail Drop

When the queue is full, packets that are trying to queue up for transmission literally have nowhere to be put! These packets are then subject to *tail drop*, which is a fancy way of saying "you're being dropped because we have no place to put you".

You know that TCP has a detection and recovery scheme when it comes to missing segments, so tail drop is no big deal, right? Quite the opposite, it's a *huge* deal.

The problem starts innocently enough, as the senders realize their TCP packets are being dropped. As we'd expect, the senders then throttle back on their transmission speed. After doing so, the senders will then gradually speed their transmission rates back up.

As multiple senders increase their tranmission rates, the queue will fill up again, and the senders will again almost simultaneously slow their tranmission rates, followed by another near-simultaneous increase. As a result of this *global synchronization,* the links are perpetually in one of two states - congested or underused. Basically, the network ends up being either hammered or not being used to its full potential, and those are both circumstances we want to avoid. One way to avoid global synchronization is though the use of Random Early Detection (RED).

Random Early Detection (RED)

At first glance, you might wonder what the difference is between RED and tail drop. If packets are being dropped randomly by RED - and from the name, obviously they are - why not just use tail drop?

RED is preferred over tail drop due to RED's ability to see that the queue is becoming congested before it's *totally* congested - that's the "early detection" part of RED. Another benefit of RED is its ability to drop packets at a higher rate as the queue length approaches its maximum.

RED will use three separate values to perform this congestion detection:

- *Minimum Queue Threshold* - This is when RED begins to drop packets.
- *Maximum Queue Threshold* - At this level, RED is dropping as many packets as it can!
- *Mark Probability Denominator* - Value used to decide by RED to decide exactly how many packets "as many packets as it can" *is*. For example, if the MPD is set to 100, one out of every 100 packets will be dropped when the queue average reaches the max queue threshold value.

Those two thresholds actually give us three possibilities regarding the number of packets in the queue:

- Between zero and the minimum threshold
- Between the minimum and maximum threshold
- Over the maximum threshold

RED has a different packet drop behavior for each of those situations.

No Drop: When the number of packets in the queue is between zero and the minimum, RED drops no packets. After all, if the queue size is below the minimum threshold, why drop packets?

Random Drop: Between the minimum and maximum thresholds, packets are randomly dropped ("early detection"). As the queue size gets closer to the maximum, RED increases the drop rate.

Full Drop: When the queue size exceeds the maximum, all newly-arrived packets are dropped until the queue size no longer exceeds that maximum. If that sounds like tail drop, that's because it is!

The *mark probability denominator* determines the probability of a packet being dropped when the number of packets in the queue reaches the maximum threshold. That might sound complicated, but it's not:

When the average queue size reaches the maximum threshold value, drop 1 of every < MDP value> packets .

If the MDP is set to 5, one out of every 5 packets will be dropped when the average queue size hits the maximum threshold. (a 20% drop probability). If the queue size exceeds the maximum threshold, Tail Drop behavior goes into effect.

Be careful - setting the MPD to 1 will result in dropping *all* excess traffic when that max threshold is reached.

RED is a major improvement over Tail Drop, but it still doesn't give us a great deal of control over the entire queueing and dropping process. With one simple word, though, we do gain that control - when we use Weighted Random Early Detection (WRED).

Weighted Random Early Detection

The "weight" in WRED refers to the amount of importance assigned to a given traffic type. IP Prec and DSCP values can be used to assign this weight, and here we'll concentrate on IP Precedence.

Since there are multiple IP Precedence values, multiple weights and thresholds can be assigned, indicating to the router that certain traffic types should be subject to packet drop before other higher-priority traffic. This allows WRED to *selectively* drop packets, rather than just the total random drop of RED.

IP Precedence values range from 0 - 7, with 7 as the highest precedence. Here's a breakdown of all IP Prec values and their common names:

7 - Network 6 - Internet 5 - Critical

4 - Flash-Override 3 - Flash 2 - Immediate

1 - Priority 0 - Routine

Enabling WRED is simple enough:

```
R3(config)#int serial0
R3(config-if)#random-detect
```

When WRED is enabled, two values are set -- the weight value used in the average queue length configuration is set to 9, and the mark probability denominator for all IP Prec values is set to 10.

What's that? I didn't give you the formula WRED uses to calculate the

average queue size? Here you are....

(old_average * (1 - 1/2^n)) + (current_queue_size * 1/2^n)

I wouldn't spend a great amount of time memorizing that formula, but there it is! "n" is set to 9 by default, and Cisco documentation recommends that you leave that at the default. So do I! But if you have a truly great reason for changing this value, you can do so with this command:

```
R3(config-if)#random-detect exponential-weighting-constant ?
  <1-16>  integer in 1..16 used in weighted average to mean 2^number
```

To assign a minimum threshold, maximum threshold, and mark probability denominator to a given IP Precedence value, use the following command:

```
R3(config-if)#random-detect precedence ?
  <0-7>  IP precedence
  rsvp   rsvp traffic

R3(config-if)#random-detect precedence 7 ?
  <1-4096>  minimum threshold (number of packets)

R3(config-if)#random-detect precedence 7 1000 ?
  <1-4096>  maximum threshold (number of packets)

R3(config-if)#random-detect precedence 7 1000 2000 ?
  <1-65536>  mark probability denominator
  <cr>

R3(config-if)#random-detect precedence 7 1000 2000 5
```

Using what we learned earlier in this chapter, at what point will traffic begin to be dropped for traffic with IP Precedence 7? Let's review....

The minimum threshold is set to 1000 packets. As long as there are fewer than 1000 packets in the queue, no packets will be dropped.

As the average queue size approaches the maximum queue size value, the mark probability denominator comes into play. I set that value to 5, meaning that one out of every 5 packets will be dropped.

When the queue length reaches 2000 packets, all new packets will be dropped.

I want to reiterate that this is strictly an example, not a config you would necessarily want to set on a production network. Cisco recommends you keep these values at their defaults, and that's an excellent idea. You should have a very good reason for changing any of these values before actually doing so!

Traffic Shaping And Policing

These two terms are often mentioned jointly, but they are far from the same thing.

Traffic shaping is more of a "friendly" policy toward excess traffic. When traffic shaping is in effect, non-conforming traffic will be buffered on its way to eventually being transmitted. Traffic shaping is particularly effective when your network has to deal with bursty traffic on a regular basis.

Traffic policing is a whole other ball game. One of two things is going to happen to non-conforming traffic when policing is in effect:

- The traffic is dropped
- The traffic is demoted in importance ("re-marked")

By the way, the technical term for non-conforming traffic is "out of profile", which results in the non-technical-sounding "OOP" acronym.

Before we look at some policing and shaping configurations, let's look at the other major differences between the two.

- Traffic policing can be applied on both an incoming and outgoing basis; traffic shaping can only be applied to outgoing traffic.

- TCP retransmissions do occur as a result of dropped packets when policing is in effect. These retransmissions occur with traffic shaping as well, but there will not be as many since traffic shaping does buffer OOP packets.

- Traffic policing does consider packet marking and will perform packet re-marking, but traffic shaping doesn't have anything to do with either packet marking or re-marking.

Traffic policing is more complex to configure than shaping, so we'll tackle that one first. A traffic policy is actually applied in a policy map, so the first part of the configuration will look very familiar! There are quite a few options in a traffic policing config, so we'll use IOS Help quite a bit in this section.

In this example, we'll assume that we want to police traffic sourced from 172.1.1.0 /24.

```
R1(config)#access-list 45 permit 172.1.1.0 0.0.0.255
R1(config)#
R1(config)#class-map GROUP45
R1(config-cmap)#match access-group 45
```

Now let's start using IOS Help to see our traffic policing options. For this

config, we'll need to define three separate values. From left to right in the *police* command, they are:

- Average transmission rate
- Normal burst size
- Excess burst size

I often mention that you should always use IOS Help when entering any value having anything to do with time or packet size. This is particularly true with traffic policing. Notice anything unusual about the below display?

```
R1(config-pmap-c)#police ?
  <8000-2000000000>  Bits per second

R1(config-pmap-c)#police 10000 ?
  <1000-512000000>  Burst bytes

R1(config-pmap-c)#police 10000 3000 ?
  <1000-512000000>  Burst bytes
```

The average rate is configured in *bits* per second, while the normal and excess burst values are configured in *bytes* per second.

In the following example, we'll define an average transmission rate of 10000 BPS, a normal burst of 3000 bytes, and an excess burst size of 4000 bytes. After doing so, we'll use IOS Help to see the options for conforming ("in-profile") traffic.

```
R1(config-pmap-c)#police 10000 3000 4000 conform-action ?
  drop                                drop packet
  exceed-action                       action when rate is within conform and
                                      conform + exceed burst
  set-clp-transmit                    set atm clp and send it
  set-discard-class-transmit          set discard-class and send it
  set-dscp-transmit                   set dscp and send it
  set-frde-transmit                   set FR DE and send it
  set-mpls-exp-imposition-transmit    set exp at tag imposition and send it
  set-mpls-exp-topmost-transmit       set exp on topmost label and send it
  set-prec-transmit                   rewrite packet precedence and send it
  set-qos-transmit                    set qos-group and send it
  transmit                            transmit packet
  <cr>
```

Quite a few options! The three we're most likely to use here are *transmit*, *set-dscp-transmit*, and *set-prec-transmit. transmit* does exactly that, and the other two options transmit the traffic after setting a specific DSCP or IP Precedence value. You have more choices with DSCP than I'll show here, but here are just a few:

```
R1(config-pmap-c)#police   10000   3000   4000   conform-action   set-dscp-
transmit ?
  <0-63>   Differentiated services codepoint value
  af11     Match packets with AF11 dscp (001010)
  af12     Match packets with AF12 dscp (001100)
```

If you're setting the IP Precedence, the choices are much simpler:

```
R1(config-pmap-c)#police 10000 3000 4000 conform-action set-prec-
transmit ?
  <0-7>  new precedence
```

Even though this may be one of the longest commands you've seen, we're not done yet! We now have to define what should happen to packets that exceed the configured rate limit but is still within the burst limits we configured.

The options should look familiar! Also note the dollar sign appearing on the left-hand side of the command - that indicates that the command is so long that not all of the characters can be shown.

```
R1(config-pmap-c)#police 10000 3000 4000 conform-action transmit ?
exceed-action  action when rate is within conform and conform + exceed
burst
  <cr>

R1(config-pmap-c)#$0 3000 4000 conform-action transmit exceed-action ?
  drop                              drop packet
  set-clp-transmit                  set atm clp and send it
  set-discard-class-transmit        set discard-class and send it
  set-dscp-transmit                 set dscp and send it
  set-frde-transmit                 set FR DE and send it
  set-mpls-exp-imposition-transmit  set exp at tag imposition and send it
  set-mpls-exp-topmost-transmit     set exp on topmost label and send it
  set-prec-transmit                 rewrite packet precedence and send it
  set-qos-transmit                  set qos-group and send it
  transmit                          transmit packet
```

Finally, we can set an action for traffic that exceeds the rate limit and burst limits combined. This *violate-action* value is not available on all IOS versions, but it's a good one to know. The *violate-action* options are the same as they were for *conform-action* and *exceed-action*, so I won't show them again, but generally you're going to configure these packets to be dropped.

```
R1(config-pmap-c)#$action transmit exceed-action set-prec-transmit 2  ?
violate-action  action when rate is greater than conform + exceed burst
```

Running *show config* verifies that we ended up with a very long command!

```
policy-map POLICEGROUP45
 class GROUP45
   police 10000 3000 4000 conform-action transmit  exceed-action set-prec-
transmit 2 violate-action drop
```

Verify the traffic policing with *show policy-map.*

```
R1#show policy-map
  Policy Map POLICEGROUP45
    Class GROUP45
     police cir 10000 bc 3000 be 4000
       conform-action transmit
```

```
      exceed-action set-prec-transmit 2
      violate-action drop
```

Not only is traffic shaping more forgiving to OOP packets than policing, it's also got few options. Traffic shaping is also configured as part of a policy map, and *shape* is the command that puts traffic shaping into effect.

```
R1(config)#policy-map POLICEGROUP45
R1(config-pmap)#class GROUP45
R1(config-pmap-c)#bandwidth 200
R1(config-pmap-c)#shape ?
 adaptive        Enable Traffic Shaping adaptation to BECN
 average         configure token bucket: CIR (bps) [Bc (bits) [Be (bits)]]
                  send out Bc only per interval
 fecn-adapt      Enable Traffic Shaping reflection of FECN as BECN
 fr-voice-adapt  Enable rate adjustment depending on voice presence
 max-buffers     Set Maximum Buffer Limit
 peak            configure token bucket: CIR (bps) [Bc (bits) [Be (bits)]]
                  send out Bc+Be per interval
```

The most common usage of the *shape* command is to set a peak value, which allows traffic in this class to go above and beyond the configured bandwidth value if there is additional bandwidth available.

Take a close look at the shape options - quite a few Frame Relay options there! While you can perform shaping on Metro Ethernet and ATM as well, most of us will configure traffic shaping primarily for Frame Relay. (Of course, that depends on your network and admin role!)

Remind me - what do we always do when configuring a command with a time or packet size value?

```
R1(config-pmap-c)#shape peak ?
<8000-154400000>  Target Bit Rate (bits per second), the value needs to be
                    multiple of 8000
```

Right! Always use IOS Help. The *peak* command is measured in bits per second, and has to be set to a multiple of 8000.

Compression Methods

Another good way to avoid congestion is to compress some of the data we're transmitting. We've got quite a few options when it comes to compression, so let's start at Layer Two.

L2 compression is actually L2 *payload* compression, an important distinction from compression methods we'll look at later in this course that only compress packet headers. From past Cisco exams, you're probably familiar with these three L2 compression methods:

- Stacker

- Predictor
- Microsoft Point-To-Point Compression

Cisco's website recommends that you choose Predictor if the congestion is due to an overloaded router, while Stacker is a good selection if limited bandwidth is the issue. Additionally, Cisco's recommendation is that you disable compression if the CPU load goes over the 40% mark.

Not all routers have hardware-based compression capabilities, but if you're working on one that does, be aware that hardware-based compression is much easier on the CPU than software-based compression. The hardware-based compression process is also faster than software-based, resulting in less delay before the compressed data is transmitted.

TCP And RTP Header Compression

Obviously, we're only compressing the header here, so CPU load really isn't an issue. What can be an issue is getting the syntax right!

Before we look at these compression methods in action, take note that these two methods compress different headers:

- RTP HC compresses the IP, RTP, and UDP headers
- TCP HC compresses the IP and TCP headers only

If you're configuring TCP HC on a non-Frame Relay interface, the config is pretty simple:

```
R2(config-if)#ip tcp header-compression ?
 ietf-format  Compressing using IETF format
 passive      Compress only for destinations which send compressed headers
  <cr>
```

You may be familiar with the *passive* option from your CCNA studies. When this option is used, headers destined for a given destination are compressed only if headers on packets from that same destination arrived in a compressed state.

If you choose IETF compression, the remote routers must be configured for IETF as well. You can't run IETF compression on Frame Relay interfaces.

There's a third option that's not available on all routers, and that is *iphc-format*. This command enables the IP Header Compression (IPHC) header compression method. When IPHC is in effect on an interface running HDLC or PPP, *RTP header compression will also be enabled*. As with IETF header compression, IPHC compression cannot run on interfaces enabled with Frame Relay.

Configuring TCP HC on interfaces running Frame Relay is a little different. I did get an interesting result when attempting to configure IETF encapsulation on an interface that was already running Frame. We're not supposed to be able to do that, right?

```
R2(config)#int s0/0/0
R2(config-if)#encap frame
R2(config-if)#ip tcp header ?
 ietf-format  Compressing using IETF format
 passive      Compress only for destinations which send compressed headers
  <cr>
```

R2(config-if)#ip tcp header ietf

I didn't get an error message when I entered that command. Interesting! I then saved the config and did a quick show config, and what did I see under the serial interface?

```
interface Serial0/0/0
 no ip address
 encapsulation frame-relay
 frame-relay lmi-type cisco
 frame-relay ip tcp header-compression
```

The router actually replaced what I entered with the correct command - *frame-relay ip tcp header-compression* - with no IETF in sight. I wouldn't count on the exam doing that for you, though!

With that command in place, *every frame map statement we add to the config will inherit that setting*. I'll add two *frame map* statements and then run our old friend *show frame map* to verify.

```
R2(config)#int s0/0/0
R2(config-if)#frame map ip 172.12.123.1 122 broadcast
R2(config-if)#frame map ip 172.12.123.2 123 broadcast

R2#show frame map
Serial0/0/0 (up): ip 172.12.123.1 dlci 122(0x7A,0x1CA0), static,
              broadcast,
              CISCO, status defined, active
              TCP/IP Header Compression (inherited), connections: 256
Serial0/0/0 (up): ip 172.12.123.2 dlci 123(0x7B,0x1CB0), static,
              broadcast,
              CISCO, status defined, active
              TCP/IP Header Compression (inherited), connections: 256
```

What if we wanted to add a third frame map statement that should not inherit the compression setting? We can use the *nocompress* option in the *frame map* statement to nullify the inheritance for that particular mapping, while still allowing the other mappings to inherit compression.

```
R2(config-if)#frame map ip 172.12.123.3 124 ?
  broadcast            Broadcasts should be forwarded to this address
  cisco                Use CISCO Encapsulation
  compress             Enable TCP/IP and RTP/IP header compression
  ietf                 Use RFC1490/RFC2427 Encapsulation
```

```
  nocompress           Do not compress TCP/IP headers
  payload-compression  Use payload compression
  rtp                  RTP header compression parameters
  tcp                  TCP header compression parameters
  <cr>

R2(config-if)#frame map ip 172.12.123.3 124 nocompress
```

Let's verify with *show frame map.*

```
R2#show frame map
Serial0/0/0 (up): ip 172.12.123.1 dlci 122(0x7A,0x1CA0), static,
              broadcast,
              CISCO, status active
              TCP/IP Header Compression (inherited), connections: 256
Serial0/0/0 (up): ip 172.12.123.2 dlci 123(0x7B,0x1CB0), static,
              broadcast,
              CISCO, status active
              TCP/IP Header Compression (inherited), connections: 256
Serial0/0/0 (up): ip 172.12.123.3 dlci 124(0x7C,0x1CC0), static,
              CISCO, status active
```

Note that the last map statement says nothing about TCP HC. The nocompress option successfully prevented that.

We can also configure a *frame map* statement to use TCP HC while *not* configuring TCP HC on the main interface. To illustrate, I'll remove TCP HC from an earlier configuration.

```
R2(config)#int s0/0/0
R2(config-if)#no frame ip tcp header
```

I'll now add two frame map statements, one configured to use TCP HC and the other will be a "normal" frame map statement.

```
R2(config)#int s0/0/0
R2(config-if)#frame map ip 172.12.123.5 155 broadcast
R2(config-if)#frame map ip 172.12.123.6 155 broadcast ?
  cisco               Use CISCO Encapsulation
  compress            Enable TCP/IP and RTP/IP header compression
  ietf                Use RFC1490/RFC2427 Encapsulation
  nocompress          Do not compress TCP/IP headers
  payload-compression  Use payload compression
  rtp                 RTP header compression parameters
 tcp                 TCP header compression parameters
  <cr>

R2(config-if)#frame map ip 172.12.123.6 155 broadcast tcp ?
  header-compression  Enable TCP/IP header compression

R2(config-if)#frame  map  ip  172.12.123.6  155  broadcast  tcp  header-
compression ?
  active       Always compress TCP/IP headers
  connections  Maximum number of compressed TCP connections
  passive      Compress for destinations sending compressed headers
  <cr>

R2(config-if)#frame  map  ip  172.12.123.6  155  broadcast  tcp  header-
compression
```

Verify with *show frame map.*

```
R2#show frame map
Serial0/0/0 (up): ip 172.12.123.5 dlci 155(0x9B,0x24B0), static,
              broadcast,
              CISCO, status active
Serial0/0/0 (up): ip 172.12.123.6 dlci 155(0x9B,0x24B0), static,
              broadcast,
              CISCO, status active
              TCP/IP Header Compression (enabled), connections: 256
```

There is a popular misconception that individual *frame map* statements cannot run TCP HC without it being enabled on the main interface. Now you know better! :)

RTP Header Compression

Realtime Transport Protocol compression is configured in much the same fashion as TCP HC. At the interface level, it's enabled with the *ip rtp header-compression* command. As with TCP HC, configuring passive RTP compression means that outbound RTP packets only have their headers compressed if that same interface is receiving RTP packets with their headers compressed.

```
R3(config)#int s0/0/0
R3(config-if)#ip rtp header-compression ?
  ietf-format       Compressing using IETF format
  iphc-format       Compress using IPHC format
  passive           Compress only for destinations which send compressed
                    headers
  <cr>

R3(config-if)#ip rtp header-compression
```

RTP header compression offers both IETF and IPHC compression options, and the rules are the same as they were with TCP HC - if one end of the channel is using either IETF or IPHC, the other end of the channel must be doing so as well.

Configuring RTP HC over Frame Relay will look much the same as TCP HC did, and the same rules apply here as well. Configuring RTP HC on the main interface will result in all frame map statements inheriting that config, as verified below by *show frame map.*

```
R3(config)#int s0/0/0
R3(config-if)#encap frame
R3(config-if)#ip rtp header-compression
R3(config-if)#frame map ip 172.12.123.1 221 broadcast
R3(config-if)#frame map ip 172.12.123.2 221 broadcast

R3#show frame map
Serial0/0/0 (up): ip 172.12.123.1 dlci 221(0xDD,0x34D0), static,
              broadcast,
```

```
              CISCO, status active
              RTP Header Compression (inherited), connections: 256
Serial0/0/0 (up): ip 172.12.123.2 dlci 221(0xDD,0x34D0), static,
              broadcast,
              CISCO, status active
              RTP Header Compression (inherited), connections: 256
```

To then configure a mapping that will not be subject to RTP HC under that interface, run *nocompress* with that frame mapping.

```
R3(config)#int s0/0/0
R3(config-if)#frame map ip 172.12.123.3 323 ?
  broadcast            Broadcasts should be forwarded to this address
  cisco                Use CISCO Encapsulation
  compress             Enable TCP/IP and RTP/IP header compression
  ietf                 Use RFC1490/RFC2427 Encapsulation
  nocompress           Do not compress TCP/IP headers
  payload-compression  Use payload compression
  rtp                  RTP header compression parameters
  tcp                  TCP header compression parameters
  <cr>

R3(config-if)#frame map ip 172.12.123.3 323 nocompress

R3#show frame map
Serial0/0/0 (up): ip 172.12.123.1 dlci 221(0xDD,0x34D0), static,
              broadcast,
              CISCO, status active
              RTP Header Compression (inherited), connections: 256
Serial0/0/0 (up): ip 172.12.123.2 dlci 221(0xDD,0x34D0), static,
              broadcast,
              CISCO, status active
              RTP Header Compression (inherited), connections: 256
Serial0/0/0 (up): ip 172.12.123.3 dlci 323(0x143,0x5030), static,
              CISCO, status active
```

To remove RTP HC from the main interface, run *no ip rtp header-compression*; to run RTP HC on an individual frame mapping while not enabling it on the main interface, use the RTP HC option on the *frame map* statement itself.

```
R3(config)#int s0/0/0
R3(config-if)#no ip rtp header-compression

R3(config-if)#frame map ip 172.12.123.4 224 broadcast rtp header-
compression

R3#show frame map
Serial0/0/0 (up): ip 172.12.123.1 dlci 221(0xDD,0x34D0), static,
              broadcast,
              CISCO, status active
Serial0/0/0 (up): ip 172.12.123.2 dlci 221(0xDD,0x34D0), static,
              broadcast,
              CISCO, status active
Serial0/0/0 (up): ip 172.12.123.3 dlci 323(0x143,0x5030), static,
              broadcast,
              CISCO, status active
Serial0/0/0 (up): ip 172.12.123.4 dlci 224(0xE0,0x3800), static,
              broadcast,
              CISCO, status active
```

```
RTP Header Compression (enabled), connections: 256
```

What Exactly Does RTP HC Compress, Anyway?

It's pretty easy to figure out what TCP HC compresses, but what about RTP HC? RTP HC will compress RTP headers, certainly - but it will also compress IP and UDP headers. Therefore, if you're using both compression methods, you're compressing both TCP and UDP headers.

RTP compression can result in quite a bit of overhead reduction. Consider those three headers and their size:

- IP Header: 20 bytes
- UDP Header: 8 bytes
- RTP Header: 12 bytes

RTP HC will result in that overall header size being reduced to anywhere from 2 to 4 bytes, depending on whose documentation you're reading.

So Which One Do I Use In My Network?

In a nutshell, use TCP HC for data transmission and RTP HC for Voice transmissions. In the past, a combination of LLQ and RTP HC has worked nicely for voice transmissions, since LLQ creates that strict priority queue for voice. Team that up with RTP HC and you've got a great way to speed up Voice transmissions.

For good ol' basic data transmissions, I personally prefer CBWFQ with TCP HC, but you can use WFQ as well.

Hot Spots And Gotchas

Traffic Shaping Vs. Traffic Policing:

Both Policing and Shaping are used to keep traffic flows from exceeding a configured rate limit, but they use very different methods of doing so.

Traffic Policing:

> Will either drop OOP packets or re-mark the traffic with a new IP Prec value, but will not delay them or "re-queue" them
>
> Can be configured to police both incoming and outgoing traffic
>
> Generally causes more TCP retransmissions than Shaping, since

Policing will generally result in more packets being dropped

Supports traffic marking

Traffic Shaping:

Delays OOP packets by queueing and then transmitting the traffic when possible rather than just dropping the packets immediately

Can be configured to shape outgoing traffic flows only

Generally fewer TCP retransmissions than Policing

Does *not* support traffic marking

WRED:

Weighted Random Early Detection detects congestion before the queue is actually full, as does RED. The big difference between the two is that WRED will *selectively* drop traffic by considering IP Precedence or DSCP values in determining which traffic should be dropped.

RED:

RED will use three separate values for packet drop:

- *Minimum Queue Threshold* - This is when RED begins to drop packets.
- *Maximum Queue Threshold* - At this level, RED is dropping as many packets as it can.
- *Mark Probability Denominator* - Value used to decide by RED to decide exactly how many packets "as many packets as it can" *is*.

RED has three packet drop modes:

No Drop: When the number of packets in the queue is between zero and the minimum, RED drops no packets. After all, if the queue size is below the minimum threshold, why drop packets?

Random Drop: Between the minimum and maximum thresholds, packets are randomly dropped ("early detection"). As the queue size gets closer to the maximum, RED increases the drop rate.

Full Drop: When the queue size exceeds the maximum, all newly-arrived packets are dropped until the queue size no longer exceeds that maximum. If that sounds like tail drop, that's because it is!

Best-practice combinations of queueing and header compression: LLQ and RTP HC for voice, CBWFQ and TCP HC for normal data

transmissions.