

Understanding Weighted Fair Queuing on ATM

Document ID: 10049

Introduction

Prerequisites

- Requirements
- Components Used
- Conventions

Network Diagram

How to Set the Transmit Ring Limit

- Impact of the Transmit Ring Limit
- Example A
- Example B

How to Calculate the Weight

How to Calculate the Scheduling Time

How WFQ Works

- What is a Particle?
- Test A
- Test B
- Stage 1
- Stage 2
- Stage 3
- Stage 4
- Summary

Related Information

Introduction

This document provides an introduction to traffic queuing that uses weighted fair queuing (WFQ) technology.

WFQ was introduced in order to enable slow-speed links, such as serial links to provide fair treatment for all types of traffic. In order to do this, WFQ classifies the traffic into different flows (also known as conversations) based on layer three and layer four information, such as IP addresses and TCP ports. It does this without the requirement of you to define access lists. This means that low-bandwidth traffic effectively has priority over high-bandwidth traffic because high-bandwidth traffic shares the transmission media in proportion to its assigned weight.

But, WFQ has certain limitations:

- It is not scalable if the flow amount increases considerably.
- Native WFQ is not available on high-speed interfaces such as ATM interfaces.

Class-based weighted fair queuing (CBWFQ) provides a solution to these limitations.

Unlike standard WFQ, CBWFQ allows you to define traffic classes. You can also apply parameters, such as bandwidth and queue-limits, to them. The bandwidth you assign to a class is used in order to calculate the weight of that class. The weight of each packet that matches the class criteria is also calculated from this. WFQ is then applied to the classes, which can include several flows, rather than the flows themselves.

Refer to these documents for more information on how to configure CBWFQ:

- Per-VC Class-Based, Weighted Fair Queuing (Per-VC CBWFQ) on Cisco 7200, 3600, and 2600 Routers
- Per-VC Class-Based, Weighted Fair Queuing on RSP-based Platforms

ATM interfaces do not support native flow-based WFQ configured directly on an interface with the **fair-queue** command. But, with the software that supports CBWFQ, you can configure flow-based WFQ within the default class, as shown in this example:

```

policy-map test
  class class-default
    fair-queue
!
interface ATMx/y.z point-to-point
ip address a.b.c.d M.M.M.M
pvc A/B
  service-policy output test

```

Prerequisites

Requirements

There are no specific requirements for this document.

Components Used

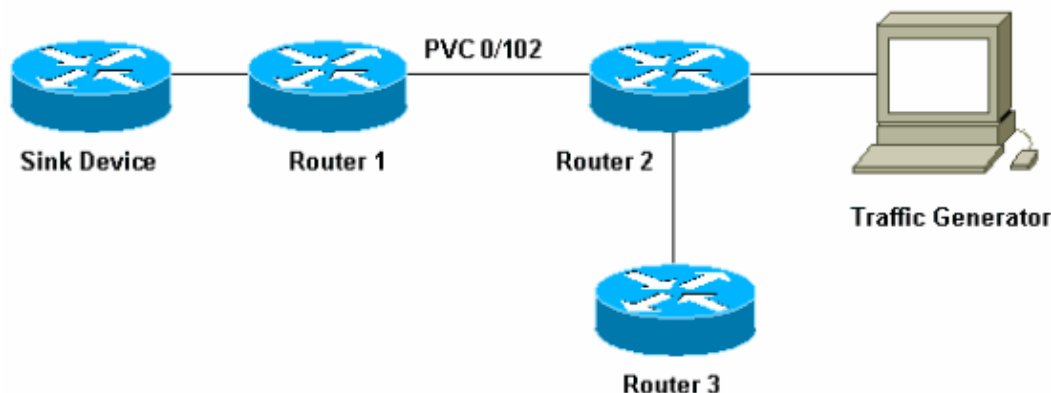
This document is not restricted to specific software and hardware versions.

Conventions

Refer to Cisco Technical Tips Conventions for more information on document conventions.

Network Diagram

Use this setup in order to illustrate how WFQ works:



In this setup, packets can be stored in one of these two queues:

- The hardware first in first out (FIFO) queue on the port adaptor and network module
- The queue in the Cisco IOS[®] software, on the router input/output [I/O] memory, where Quality of Service (QoS) features such as CBWFQ can be applied

The FIFO queue on the port adaptor stores the packets before they are segmented into cells for transmission. When this queue is full, the port adaptor or network module signals to the IOS software that the queue is congested. This mechanism is called back-pressure. On the receipt of this signal, the router stops to send packets to the interface FIFO queue and stores the packets in the IOS software until the queue is uncongested again. When the packets are stored in IOS, the system can apply QoS.

How to Set the Transmit Ring Limit

One problem with this queuing mechanism is that, the bigger the FIFO queue on the interface, the longer the delay before packets at the end of this queue can be transmitted. This can cause severe performance problems for delay-sensitive traffic such as voice traffic.

The permanent virtual circuit (PVC) **tx-ring-limit** command enables you to reduce the size of the FIFO queue.

```
interface ATMx/y.z point-to-point
 ip address a.b.c.d M.M.M.M
 PVC A/B
   tx-ring-limit <size>

 service-policy output test
```

The *<size>* you can specify here is either a number of packets, for Cisco 2600 and 3600 routers, or quantity of particles, for Cisco 7200 and 7500 routers.

The reduction of the size of the transmit ring has two benefits:

- It reduces the amount of time packets wait in the FIFO queue before it is segmented.
- It accelerates the use of the QoS in the IOS software.

Impact of the Transmit Ring Limit

Look at the impact of the transmit ring limit that uses the setup shown in the previous network diagram. Assume these:

- The traffic generator sends traffic (1500 byte packets) to the sink device, and this traffic overloads the PVC 0/102 between router1 and router2.
- Router3 tries to ping router1.
- WFQ is enabled on router2.

Look at two configurations that uses different transmit ring limits in order to see the impact this has.

Example A

In this example, you set the transmit ring to three (**tx-ring-limit=3**). This is what you see when you ping router1 from router3:

```
pound#ping ip
Target IP address: 6.6.6.6
Repeat count [5]: 100000
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 100000, 100-byte ICMP Echos to 6.6.6.6, timeout is 2 seconds:
```



```
Conversations 2/3/16 (active/max active/max total)
Reserved Conversations 0/0 (allocated/max allocated)
```

```
(depth/weight/discards/tail drops/interleaves) 1/32384/0/0/0
Conversation 2, linktype: ip, length: 58
source: 8.0.0.1, destination: 6.6.6.6, id: 0x2DA1, ttl: 254, prot: 1

(depth/weight/discards/tail drops/interleaves) 64/32384/1505776/0/0
Conversation 15, linktype: ip, length: 1494
source: 7.0.0.1, destination: 6.6.6.6, id: 0x0000, ttl: 63, prot: 255
```

When you use WFQ, you can calculate the weight of each conversation with the use of this formula:

- **weight=32384/(precedence+1)** – for Cisco IOS Software Release 12.0(5)T and later.
- **weight=4096/(precedence+1)** – for Cisco IOS Software Releases prior to 12.0(5)T.

How to Calculate the Scheduling Time

You can now use these weights in order to calculate the scheduling time of each packet, when the packet is forwarded from the IOS queue to the port adaptor or network module FIFO queue.

You can calculate the output scheduling time with the use of this formula, where **queue_tail_time** is the current scheduling time:

output scheduling time= queue_tail_time + pktsize*weight

How WFQ Works

This section explains how WFQ works. The principle of WFQ is that packets with a small weight, or small packets, should get priority when they are sent.

Create a flow that comprises ten packets large packets and four smaller packets (of 82 bytes) that uses a traffic generator in order to verify this.

In this example, router2 is a Cisco 7200 router with a PA-A3 (ATM Port Adapter). This is important because the size of the output FIFO queue on the port adaptor is expressed in particles and not in packets. See the [What is a Particle?](#) section for further information.

What is a Particle?

Instead of the allocation of one piece of contiguous memory for a buffer, particle buffering allocates discontinuous (scattered) pieces of memory, called particles, and then links them together in order to form one logical packet buffer. This is called a particle buffer. In such a scheme, a packet can then be spread across multiple particles.

In the 7200 router, the particle size is 512 bytes.

Use the **show buffers** command in order to verify whether Cisco 7200 routers use particles:

```
router#show buffers
[snip]
Private particle pools:
FastEthernet0/0 buffers, 512 bytes (total 400, permanent 400):
  0 in free list (0 min, 400 max allowed)
  400 hits, 0 fallbacks
  400 max cache size, 271 in cache
```



```
.Nov 7 15:39:15.208: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:39:15.208: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
```

!--- Congestion occurs at this point.

```
.Nov 7 15:39:15.512: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:39:15.516: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:39:15.644: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:39:15.644: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:39:15.776: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:39:15.776: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:39:15.904: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:39:15.904: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:39:16.384: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:39:16.384: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:39:16.860: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:39:16.860: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:39:17.340: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:39:17.340: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:39:17.816: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:39:17.820: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:39:18.296: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:39:18.296: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:39:18.776: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:39:18.776: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
```

You can see the four packets of 482 bytes sent before the 82-byte packets, which should normally get the priority. This is why this happens.

Since the burst is primarily composed of ten 482-byte packets, these reach the router first, followed by the 82-byte packets. Since the 482-byte packets arrive at a time when there is no congestion, as there is no traffic, one packet is immediately queued to the port adaptor segmentation and reassembly (SAR) to be chunked into cells and sent on the wire. In other words, the transmit ring is still empty.

You can calculate that the time needed to send one 482-byte packet is greater than the time needed for the traffic generator in order to send the total burst. You can therefore assume that, when the first 482-byte packet is queued to the port adaptor, more 482-byte packets of the burst are already present in the router. Hence, more 482-byte packets can be queued to the transmit ring. Three more packets of 482 bytes are queued with the use of the three free particles present there.

Note: Packets are queued into the transmit ring as soon as there is a free particle, even if they need more than one particle to be stored.

At this point, there is congestion, since the three particles are full. Therefore, queuing starts in IOS. When the four 82-byte packets finally reach the router, there is congestion. These four packets are queued and WFQ is used on the two flows. Look at the ATM queue that uses the **show queue ATM** command in order to see this:

```
router2#show queue ATM 4/0.102 vc 0/102
Interface ATM4/0.102 VC 0/102
Queuing strategy: weighted fair
Total output drops per VC: 0
Output queue: 10/512/64/0 (size/max total/threshold/drops)
  Conversations 2/4/16 (active/max active/max total)
  Reserved Conversations 0/0 (allocated/max allocated)

(depth/weight/total drops/no-buffer drops/interleaves) 4/32384/0/0/0
Conversation 6, linktype: ip, length: 82
source: 7.0.0.200, destination: 6.6.6.6, id: 0x0000, ttl: 63, prot: 255
```

```
(depth/weight/total drops/no-buffer drops/interleaves) 6/32384/0/0/0
Conversation 15, linktype: ip, length: 482
source: 7.0.0.1, destination: 6.6.6.6, id: 0x0000, ttl: 63, prot: 255
```

You can see in the debugs that the first four packets of 482 bytes are followed by the 82-byte packets. These small packets go out of the router before the large packets. This indicates that, as soon as congestion occurs, small packets get priority over large packets.

Use the weight and scheduling time formulas given in the Calculating the Weight section in order to verify this.

Stage 2

If you increase the transmit ring limit to five and the large packets are 482 bytes then, in accordance to the previous output, you should see six packets of 482 bytes before congestion occurs, followed by four packets of 82 bytes, then another four packets of 482 bytes:

```
.Nov 7 15:49:57.365: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:49:57.365: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:49:57.841: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:49:57.845: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:49:58.321: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:49:58.321: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:49:58.797: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:49:58.801: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:49:59.277: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:49:59.277: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:49:59.757: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:49:59.757: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:49:59.973: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:49:59.973: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:50:00.105: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:50:00.105: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:50:00.232: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:50:00.232: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:50:00.364: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:50:00.364: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown pro
.Nov 7 15:50:00.840: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:50:00.844: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:50:01.320: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:50:01.320: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:50:01.796: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:50:01.800: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
.Nov 7 15:50:02.276: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, rcvd 4
.Nov 7 15:50:02.276: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 482, unknown prot
```

As you can see here, this is indeed what happens.

Stage 3

The particle size is 512 bytes. Therefore, if the transmit ring is expressed in particles, and you use packets slightly bigger than the particle size, each one takes two particles. This is illustrated by the use of packets of 582 bytes and a transmit ring of three. With these parameters, you should see three packets of 582 bytes. One is sent with no traffic on the ATM interface, that leaves three particles free. Therefore, two more packets can be queued, followed by four packets of 82 bytes and then seven packets of 582 bytes:

```
.Nov 7 15:51:34.604: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:34.604: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:35.168: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:35.168: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
```

```

.Nov 7 15:51:35.732: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:35.736: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:35.864: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:51:35.864: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 7 15:51:35.996: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:51:35.996: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 7 15:51:36.124: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:51:36.124: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 7 15:51:36.256: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 7 15:51:36.256: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 7 15:51:36.820: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:36.820: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:37.384: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:37.388: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:37.952: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:37.952: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:38.604: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:38.604: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:39.168: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:39.168: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:39.732: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:39.736: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot
.Nov 7 15:51:40.300: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, rcvd 4
.Nov 7 15:51:40.300: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 582, unknown prot

```

Stage 4

Take a packet size of 1482 (three particles) and define a transmit ring of five. If the transmit ring is defined in particles, you see something similar:

- One packet transmitted immediately
- One packet that takes three of the five particles
- One packet queued because two particles are free

```

.Nov 8 07:22:41.200: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:41.200: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:42.592: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:42.592: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:43.984: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:43.984: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:44.112: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 8 07:22:44.112: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 8 07:22:44.332: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 8 07:22:44.332: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 8 07:22:44.460: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 8 07:22:44.460: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 8 07:22:44.591: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, rcvd 4
.Nov 8 07:22:44.591: IP: s=7.0.0.200 (FastEthernet0/1), d=6.6.6.6, Len 82, unknown prot
.Nov 8 07:22:45.983: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:45.983: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:47.371: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:47.375: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:48.763: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:48.767: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:50.155: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:50.155: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:51.547: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:51.547: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:53.027: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:53.027: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot
.Nov 8 07:22:54.415: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, rcvd 4
.Nov 8 07:22:54.419: IP: s=7.0.0.1 (FastEthernet0/1), d=6.6.6.6, Len 1482, unknown prot

```

Summary

From the tests carried out, you can conclude these:

- On slow PVCs without WFQ, bulk traffic impacts small traffic, such as pings being stalled until WFQ is enabled.
 - The size of the transmit ring (tx-ring-limit) determines how quickly the queuing mechanism starts to do its job. You can see the impact of this with the increase of the ping RTT when the transmit ring limit increases. Therefore, if WFQ or LLQ needs to be implemented, it makes sense to reduce the transmit ring limit.
 - WFQ that uses CBWFQ indeed prioritizes small traffics over bulk traffic.
-

Related Information

- [ATM Technology Support Pages](#)
 - [Congestion Management Overview](#)
 - [Technical Support & Documentation – Cisco Systems](#)
-

[Contacts & Feedback](#) | [Help](#) | [Site Map](#)

© 2007 – 2008 Cisco Systems, Inc. All rights reserved. [Terms & Conditions](#) | [Privacy Statement](#) | [Cookie Policy](#) | [Trademarks of Cisco Systems, Inc.](#)

Updated: Oct 05, 2006

Document ID: 10049
