

**CBWFQ**  
**and**  
**max-reserved-bandwidth**

**Pavel Bykov, ALEF NULA, a.s.**  
**Group Study discussion**

# Contents

<b>INTRODUCTION</b> .....	<b>3</b>
<b>1 TEST SETUP</b> .....	<b>3</b>
1.1 HARDWARE LIST .....	3
1.2 PICTURE .....	3
1.3 TOPOLOGY .....	4
1.4 GENERAL LOGIC .....	4
1.5 BASE RELEVANT CONFIG OF R4.....	4
<b>2 GETTING STARTED</b> .....	<b>5</b>
2.1 MEASUREMENT OF MEASUREMENT .....	5
<b>3 FIRST MEASUREMENTS</b> .....	<b>13</b>
3.1 SIMPLE TEST.....	13
<b>4 MAX-RESERVED-BANDWIDTH</b> .....	<b>15</b>
4.1 THE REASON FOR THIS DOCUMENT.....	15
<b>5 CONCLUSION</b> .....	<b>22</b>
5.1 .....	22
<b>6 PRE 12.4.20T</b> .....	<b>22</b>
6.1 CBWFQ MECHANISM AND CLASS-DEFAULT STARVATION. ....	22
<b>7 POST 12.4.20T CBWFQ NOTES</b> .....	<b>26</b>
7.1 RESERVATION TYPES.....	26
7.2 BANDWIDTH REMAINING PERCENT.....	26
7.3 SHOW QUEUE.....	26
7.4 DEBUG QOS.....	27

# Introduction

Cisco's opinion on "max-reserved-bandwidth " command is widely known from CCO documentation

## **max-reserved-bandwidth**

To change the percent of interface bandwidth allocated for Resource Reservation Protocol (RSVP), class-based weighted fair queueing (CBWFQ), low latency queueing (LLQ), IP RTP Priority, Frame Relay IP RTP Priority, and Frame Relay PVC Interface Priority Queueing (PIPQ), use the max-reserved bandwidth command in interface configuration mode. To restore the default value, use the no form of this command.

But is this really true? My perception is very different and has always been that "max-reserved-bandwidth" command is merely a configuration constraint, where administrator is not able to configure more than the set value, but not limitation in any technical sense.

The following testing will shed some light on the command, CBWFQ and queuing itself, but more importantly will prove one of us (almost) wrong.

## 1 Test setup

The following describes test setup. There is nothing fancy here, but the information is important for orientation or to find inconsistencies.

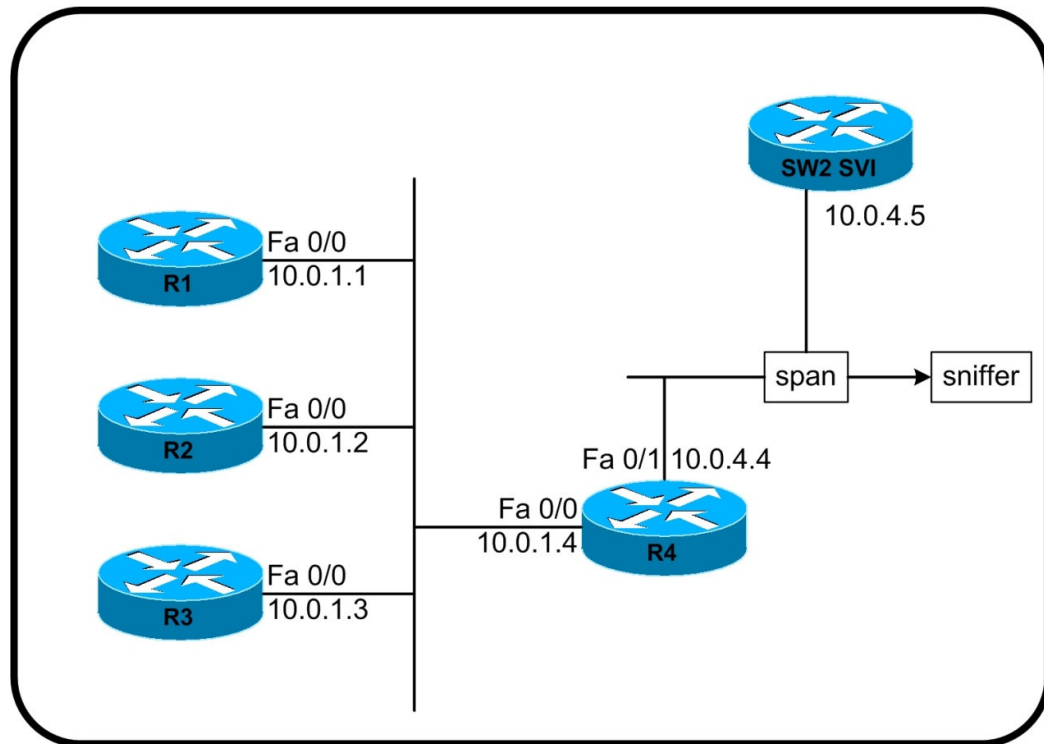
### 1.1 Hardware list

4x Identical 2811  
2x WS-C2960-24TT-L

### 1.2 Picture



### 1.3 Topology



### 1.4 General Logic

R1, R2, and R3 are all generating traffic using following command:

```
ping 10.0.4.5 size 1500 repeat 1000000000 timeout 0
```

This will ensure around 10 Mbps of traffic for as long as needed. On the other side – the 10.0.4.5 – is an SVI of SW2. Not to load the CPU of the switch, there is simply an ACL that drops all traffic. On SW2 there is a SPAN session, that copies all traffic and sends it to the Linux box with Wireshark.

```
SW2#sh mon sess 1
Session 1
-----
Type                : Local Session
Source Ports        :
  Both              : Fa0/1
Destination Ports   : Fa0/24
Encapsulation       : Replicate
Ingress             : Disabled
```

Wireshark will be used for graphing and to complement IOS command outputs, that do not always provide enough relevant information.

### 1.5 Base relevant config of R4

```
!
access-list 1 permit 10.0.1.1
access-list 2 permit 10.0.1.2
access-list 3 permit 10.0.1.3
```

```

!
class-map match-all R1
match access-group 1
class-map match-all R2
match access-group 2
class-map match-all R3
match access-group 3
!
policy-map INPUT-MONITOR
class R1
class R2
class R3
!
interface FastEthernet0/0
ip address 10.0.1.4 255.255.255.0
load-interval 30
shutdown
duplex auto
speed auto
service-policy input INPUT-MONITOR
!
interface FastEthernet0/1
ip address 10.0.4.4 255.255.255.0
load-interval 30
shutdown
duplex full
speed 10
!

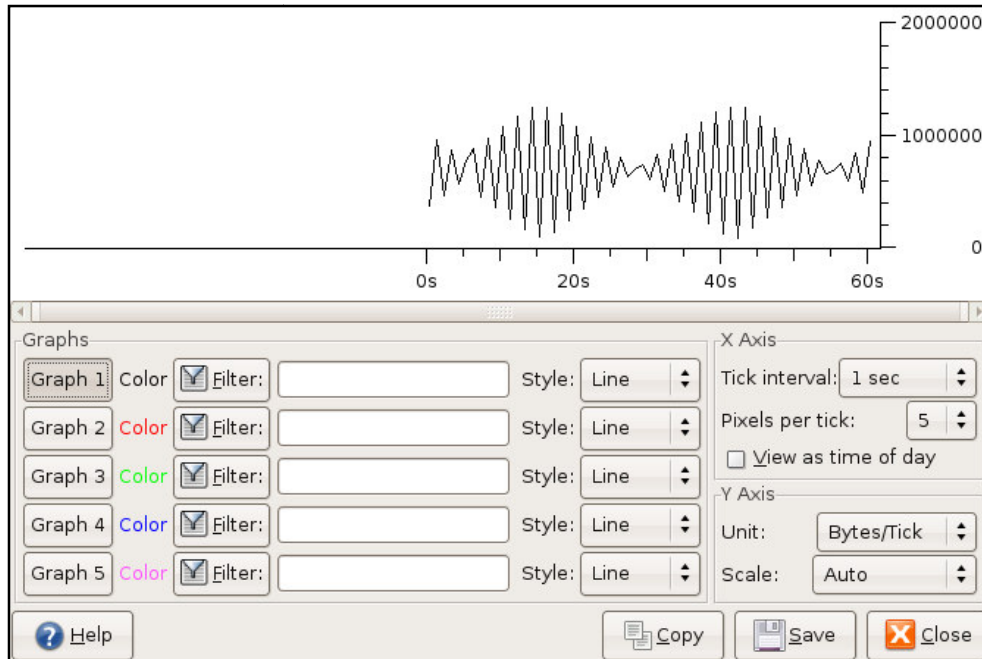
```

INPUT-MONITOR is a simple policy-map applied on input direction that will provide measuring to confirm information or find inconsistencies.

## 2 Getting Started

### 2.1 Measurement of measurement

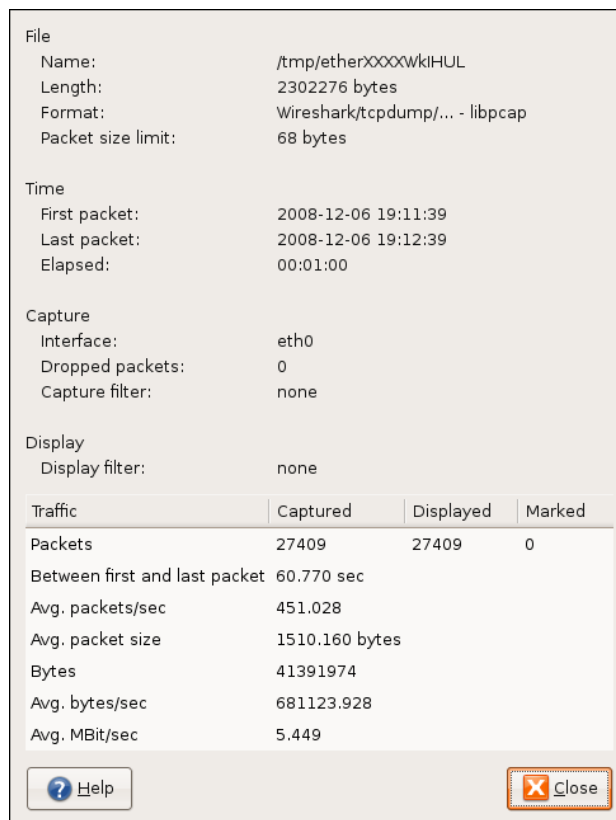
To get results, REAL results, we will first need to get consistency in our testing, meaning we need to find what predictably works in the lab, not on the paper. For now we know that R1, R2 and R3 are all capable of generating around 10 Mbps of traffic. But do those packets really get to the destination? The question may seem too simple: if 10Mbps comes in, 10Mbps comes out. In reality, this cannot be further from a truth. Let's do a simple test to find out. Without applying any additional configuration at R4, we start generating pings from R1 to Receiver at 10Mbps. We start wireshark sniffer and sniff for 60 seconds. Now let's look at I/O graph output.



**Figure 1**

Note: the units are Bytes/Tick not Bits/Tick, so 10Mbps is 1.25MB. Other figures will correctly display bits/sec

Hmm... Does that look like what we expected? Zig-Zagging well below 10 Mbps? Not exactly you say? Well, let's look at the summary:



**Figure 2**

Take a closer look at the last line: "Avg.MBit/sec 5.449 Mbps". Now where the error might be in that? Maybe there was some other traffic that got in the way? Let's look at the conversations

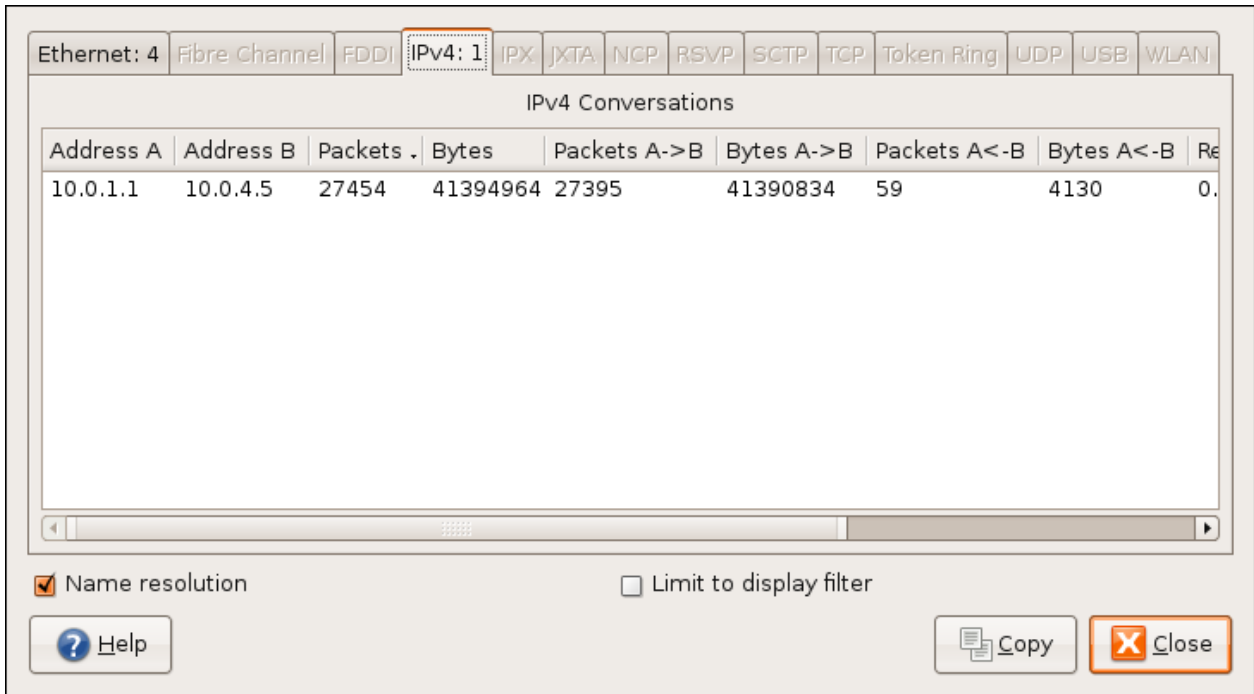


Figure 3

Nope, only we see only data from R1 (10.0.1.1) to Receiver (10.0.4.5). Let's look at R4 interfaces. R4's input interface Fa0/0 reports this:

```
R4#sh int fa 0/0
FastEthernet0/0 is up, line protocol is up
  Hardware is MV96340 Ethernet, address is 0019.e764.ec18 (bia 0019.e764.ec18)
  Internet address is 10.0.1.4/24
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
    reliability 255/255, txload 1/255, rxload 25/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:02, output 00:00:00, output hang never
  Last clearing of "show interface" counters 00:07:35
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  30 second input rate 10190000 bits/sec, 841 packets/sec
  30 second output rate 0 bits/sec, 1 packets/sec
  386495 packets input, 585144430 bytes
    Received 8 broadcasts, 0 runts, 0 giants, 0 throttles
  0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
  0 watchdog
  0 input packets with dribble condition detected
  493 packets output, 36377 bytes, 0 underruns
  0 output errors, 0 collisions, 0 interface resets
  0 unknown protocol drops
  0 babbles, 0 late collision, 0 deferred
  0 lost carrier, 0 no carrier
  0 output buffer failures, 0 output buffers swapped out
```

This looks correct. We were expecting to see about 10 Mbps of traffic, and that is what we are seeing. But look at statistics of output interface Fa 0/1:

```
R4#sh int fa 0/1
FastEthernet0/1 is up, line protocol is up
  Hardware is MV96340 Ethernet, address is 0019.e764.ec19 (bia 0019.e764.ec19)
  Internet address is 10.0.4.4/24
  MTU 1500 bytes, BW 10000 Kbit/sec, DLY 1000 usec,
    reliability 255/255, txload 139/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 10Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:54:00, output 00:00:00, output hang never
  Last clearing of "show interface" counters 00:07:40
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 184465
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  30 second input rate 0 bits/sec, 1 packets/sec
  30 second output rate 5489000 bits/sec, 453 packets/sec
    442 packets input, 30940 bytes
      Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
      0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
      0 watchdog
      0 input packets with dribble condition detected
    206475 packets output, 312529342 bytes, 0 underruns
      0 output errors, 0 collisions, 0 interface resets
      0 unknown protocol drops
      0 babbles, 0 late collision, 0 deferred
      0 lost carrier, 0 no carrier
      0 output buffer failures, 0 output buffers swapped out
```

The output rate is consistent from what we saw from our Wireshark report, but where are the other 4.5 Mbps? Almost half of our flow just got lost in the router somewhere, but where? For this, we need to quickly look at the technology itself. The input interface Fa 0/0 is runs at 100Mbps/Full Duplex, but the output interface Fa0/1 is 10Mbps/Full Duplex. The interface itself can only send and receive data at speeds that it was rated at. Simply said, it can only have two states:

**0 : OFF**  
**1 : ON**

Ah, the binary logic of it all. Of course, this is seen only at the microsecond level. What we see on all the counters are statistics over time – how much data was transferred over 5 minutes, 1 minute, or whatever we set our “load-interval” to. Meaning if we had 100Gbps interface and sent 37.5 Mega bytes of traffic in an instant, and for the next 29.9 seconds we wouldn't send anything, at the end of 30 seconds our average would be:  $37.5 \times 8 = 300$  Mega bits over 30 seconds, or 10Mbps. Will my puny 10Mbps interface be able to keep up with this rate if in the next 30 seconds we would send another 37.5 Mega bytes of data?

The answer is the key to our lost 4.5 Mbps: It depends on buff depth. We would need a buffer of about 37.5 megs to store the data and gradually output the packets from buffer to interface. 100Gbps goliath is able to send 37.5 MB of data in 3 milliseconds. At the same time, 10Mbps interface is able to send 3.75KB, and the rest has to be buffered. First, the Tx Ring (hardware output queue) is filled, and then the software queue starts to fill up. Once all buffers are full, the rest is dropped due to buffer overflow. Tx Ring is always hardware dependant, and it's always FIFO. All we can change is its length. (its default packet\* depth is hardware dependant)

\*Also, what is meant by "packet" when we talk about buffer depths depends on hardware. In ATM, for example, packet is 580 byte particle: [http://www.cisco.com/en/US/tech/tk39/tk824/technologies\\_tech\\_note09186a00800fbafc.shtml](http://www.cisco.com/en/US/tech/tk39/tk824/technologies_tech_note09186a00800fbafc.shtml)  
Tx Ring can be displayed using show controllers command.

Software queue is 40 packets long by default, so if we ignore Tx Ring for simplicity's sake, our 10Mbps interface in this case will be able to send 3.75KB + 40 packets, which is 3.75KB + 40x1518B = 64.47 KB. In 30 seconds, if we would look at statistics of our 100Gbps interface we would see 10Mbps but on our 10Mbps interface we would see only 64.47 \* 8 = 515.76 Kb / 30 s = 17.192 Kbps. This is happening on R4 in our case, but with less dramatic consequences.

We can dramatically affect results if we increase our buffer depth. For this extreme case of 100Gbps VS 10Mbps speed mismatch, we would need queue with enormous depth. For our lab case with only 100Mbps VS 10Mbps speed mismatch we would need smaller buffers. Because we really don't know how bursty our ping senders will get, we need to set buffers to be able to always cover for the difference between 100Mbps and 10Mbps over 1 second interval at packet size of 250 Bytes. Let's calculate: 100Mb - 10 Mb = 90Mb / 8 = 11.25 MB / 250B = 45000 packets... Wow, that's a large buffer. In fact, it's way over our platform capabilities.

#### Interface level:

```
R4(config-if)#int fa 0/1
R4(config-if)#hold-queue ?
<0-4096> Queue length
```

#### Policy-map level:

```
R4(config)#policy-map T1
R4(config-pmap)#class R1
R4(config-pmap-c)#queue-limit ?
<1-8192000> in bytes, <1-3400> in ms, <1-32768> in packets by default
```

Now we have two options. Either to go with lesser buffers and pray that our traffic generators would not be too bursty, or to reduce output interface speed on R1, R2 and R3 to 10 Mbps and therefore shape-out the traffic there, so we can work with very predictable values. I decided to go for option 2 and set Fa0/0 interface of R1, R2 and R3 to 10Mbps/Full Duplex, including the switch ports on the other side.

But first, let's just quickly check the theory described above. First, we will try to increase buffer depth directly at interface level:

```
R4(config)#int fa 0/1
R4(config)#hold-queue 4096 out
```

So how does this improve situation?

```
R4#sh int fa 0/1
FastEthernet0/1 is up, line protocol is up
  Hardware is MV96340 Ethernet, address is 0019.e764.ec19 (bia 0019.e764.ec19)
  Internet address is 10.0.4.4/24
  MTU 1500 bytes, BW 10000 Kbit/sec, DLY 1000 usec,
    reliability 255/255, txload 219/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 10Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:47, output 00:00:00, output hang never
  Last clearing of "show interface" counters 00:00:44
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 2226/4096 (size/max)
  30 second input rate 0 bits/sec, 1 packets/sec
```

```
30 second output rate 8627000 bits/sec, 715 packets/sec
44 packets input, 3080 bytes
Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
0 watchdog
0 input packets with dribble condition detected
36115 packets output, 54672294 bytes, 0 underruns
0 output errors, 0 collisions, 0 interface resets
0 unknown protocol drops
0 babbles, 0 late collision, 0 deferred
0 lost carrier, 0 no carrier
0 output buffer failures, 0 output buffers swapped out
```

Better, but no cigar. Here you can feel how buffers are important. Even 4096 packets aren't able to compensate for speed mismatch of 10:1, even though the overall long-term flow speed is very small. You can now see why we NEED QoS really end-to-end, even on the LANs. Because if we have just 10 users on 100Mbps ports with one 100Mbps uplink, look how many packets can get lost on that uplink interface, even if overall long-term speed will look O.K. And recommended uplink over subscription is 20:1, which in practice can be anywhere from that to 200:1. So let's try it now with policy map, which offered us to set larger buffers.

```
policy-map GS
class R1
  bandwidth percent 5
  queue-limit 32768 packets
```

And this resulted in the following:

```
R4#sh int fa 0/1 | in rate [0-9]
30 second input rate 0 bits/sec, 1 packets/sec
30 second output rate 9832000 bits/sec, 812 packets/sec
```

It's almost 10 Mbps... almost. Let's look at the input

```
R4#sh int fa 0/0 | in rate [0-9]
30 second input rate 10482000 bits/sec, 866 packets/sec
30 second output rate 0 bits/sec, 1 packets/sec
```

Output is constantly over 10 Mbps. It seems unbelievable, but even gigantic queue of 32768 packets cannot hold the 10:1 speed mismatch, and therefore we cannot completely load the interface, and some of the time our Fa 0/1 is still in the OFF state. Packets are still being dropped! Now let's see if policy-map will show us the expected drop rate.

```
R4#sh policy-map int fa 0/1
FastEthernet0/1

Service-policy output: GS

Class-map: R1 (match-all)
 72485 packets, 109742290 bytes
 30 second offered rate 9928000 bps, drop rate 236000 bps
Match: access-group 1
Queueing
queue limit 32768 packets
(queue depth/total drops/no-buffer drops) 3417/1214/1214
(pkts output/bytes output) 71271/107904294
bandwidth 5% (500 kbps)
```

```

Class-map: class-default (match-any)
  10 packets, 917 bytes
  30 second offered rate 0 bps, drop rate 0 bps
  Match: any

  queue limit 64 packets
  (queue depth/total drops/no-buffer drops) 0/0/0
  (pkts output/bytes output) 10/917

```

The drop rate is very low but it is there as expected. Some uninteresting packets like CDP went through class-default as expected as well, and the rest is in R1 class.

It is obvious now why we need to reduce the speed on interfaces of R1, R2 and R3. But it is not that simple and will not completely solve our problem. Why? Because that would just drop packets at output interface of those routers and not at our router. Burstiness that we saw originated at CPU level of the router, not on the interface level. After generating traffic at R1 for a couple of minutes, we can see this at R1's output interface:

```

R1#sh int fa 0/0
FastEthernet0/0 is up, line protocol is up
  Hardware is MV96340 Ethernet, address is 0019.56d9.2530 (bia 0019.56d9.2530)
  Internet address is 10.0.1.1/24
  MTU 1500 bytes, BW 10000 Kbit/sec, DLY 1000 usec,
    reliability 255/255, txload 145/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 10Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:09, output 00:00:02, output hang never
  Last clearing of "show interface" counters 00:03:53
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 37804
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 5703000 bits/sec, 458 packets/sec
    103 packets input, 9068 bytes
    Received 10 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog
    0 input packets with dribble condition detected
  41244 packets output, 62398634 bytes, 0 underruns
    0 output errors, 0 collisions, 0 interface resets

```

So I quickly wrote a policy-map that will allow all three routers to send most data at 10Mbps speeds and applied it on R1, R2 and R3:

```

policy-map BUFFER
  class class-default
    bandwidth percent 100
    queue-limit 8192000 bytes
  int fa 0/0
  service-policy out BUFFER

```

Granted, this does not generate me full 10Mbps (meaning a bit over 10Mbps), but it is very measurable traffic. Look how consistent it is after 5 minutes of running with default queuing mechanisms on R4

#### R4, Input Interface

```

R4#sh int fa 0/0 | in rate [0-9][Qq]ueue
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  30 second input rate 9844000 bits/sec, 812 packets/sec
  30 second output rate 0 bits/sec, 1 packets/sec

```

#### R4, Output Interface

```
R4#sh int fa 0/1 | in rate [0-9][Qq]ueue
Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
Queueing strategy: fifo
Output queue: 0/40 (size/max)
30 second input rate 0 bits/sec, 1 packets/sec
30 second output rate 9844000 bits/sec, 812 packets/sec
```

To confirm our findings, we finally run wireshark and collect data for 60 seconds. First is the data over time with the same graph precision we used last time

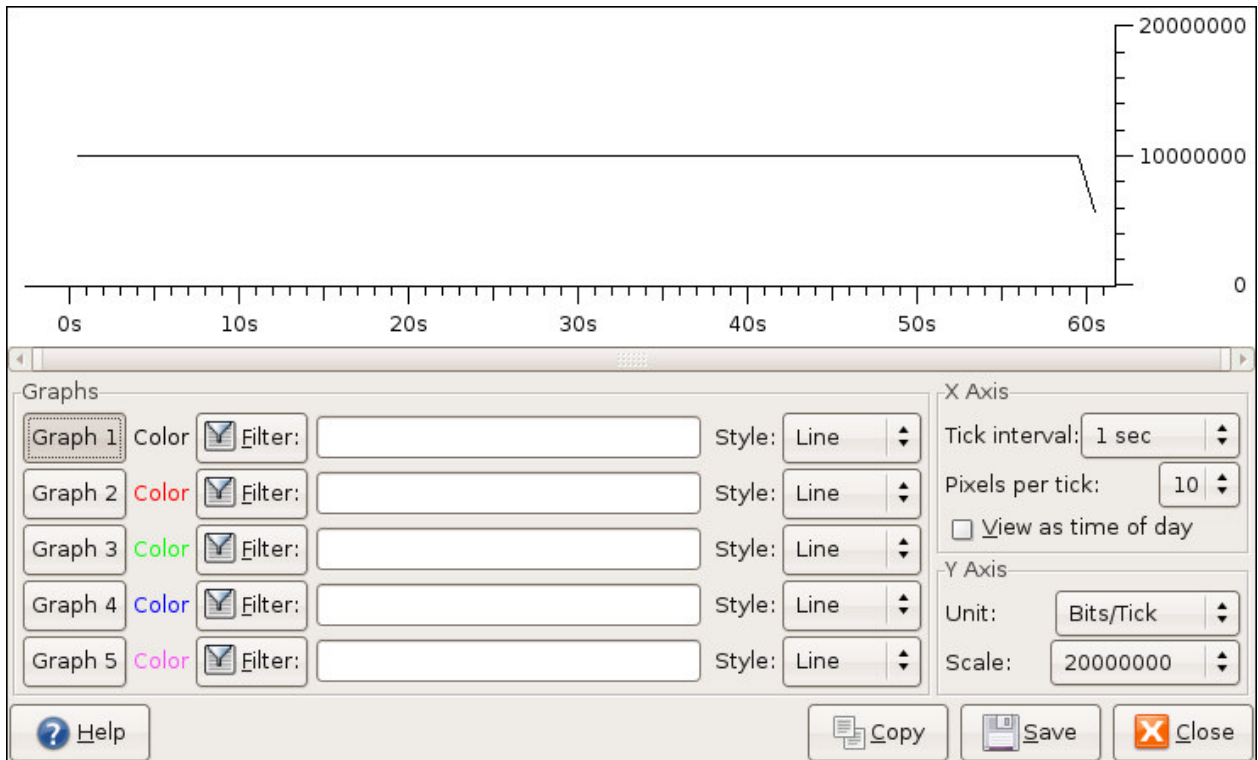


Figure 4

Wow, that's really straight. It looks almost like it was drawn. I better save sniffer output for later, just in case someone accuses me of drawing it in Gimp. (libpcap name 004)

Next, summary statistics will show us the bit rate that wireshark measured over 60 seconds



Figure 5

This figure really pleased me. 9.845 Mbps throughput when our interface counters show 9.844 Mbps is an error of 0.01% which is very good by any standards.

Now we can finally start measuring.

## 3 First Measurements

### 3.1 Simple test

Let's start with a simple one. Equally divide bandwidth between R1, R2 and R3 and measure on the output.

```
R4 Configuration
policy-map T3
```

```
class R1
  bandwidth percent 33
  queue-limit 100 packets
class R2
  bandwidth percent 33
  queue-limit 100 packets
class R3
  bandwidth percent 33
  queue-limit 100 packets
```

Now we measure the output. Policy map precision is phenomenal

```
R4#sh policy-map int fa 0/1
FastEthernet0/1

Service-policy output: T3

Class-map: R1 (match-all)
 460497 packets, 697192458 bytes
 30 second offered rate 9843000 bps, drop rate 6562000 bps
Match: access-group 1
Queueing
queue limit 100 packets
(queue depth/total drops/no-buffer drops) 99/306999/0
(pkts output/bytes output) 153499/232397486
bandwidth 33% (3300 kbps)

Class-map: R2 (match-all)
 460497 packets, 697192458 bytes
 30 second offered rate 9843000 bps, drop rate 6562000 bps
Match: access-group 2
Queueing
queue limit 100 packets
(queue depth/total drops/no-buffer drops) 100/307000/0
(pkts output/bytes output) 153498/232395972
bandwidth 33% (3300 kbps)

Class-map: R3 (match-all)
 460495 packets, 697189430 bytes
 30 second offered rate 9843000 bps, drop rate 6562000 bps
Match: access-group 3
Queueing
queue limit 100 packets
(queue depth/total drops/no-buffer drops) 100/306998/0
(pkts output/bytes output) 153499/232397486
bandwidth 33% (3300 kbps)

Class-map: class-default (match-any)
 68 packets, 7097 bytes
 30 second offered rate 0 bps, drop rate 0 bps
Match: any

queue limit 64 packets
(queue depth/total drops/no-buffer drops) 0/0/0
(pkts output/bytes output) 68/7097
```

Wireshark confirms this finding. CBWFQ works with nearly machine precision.

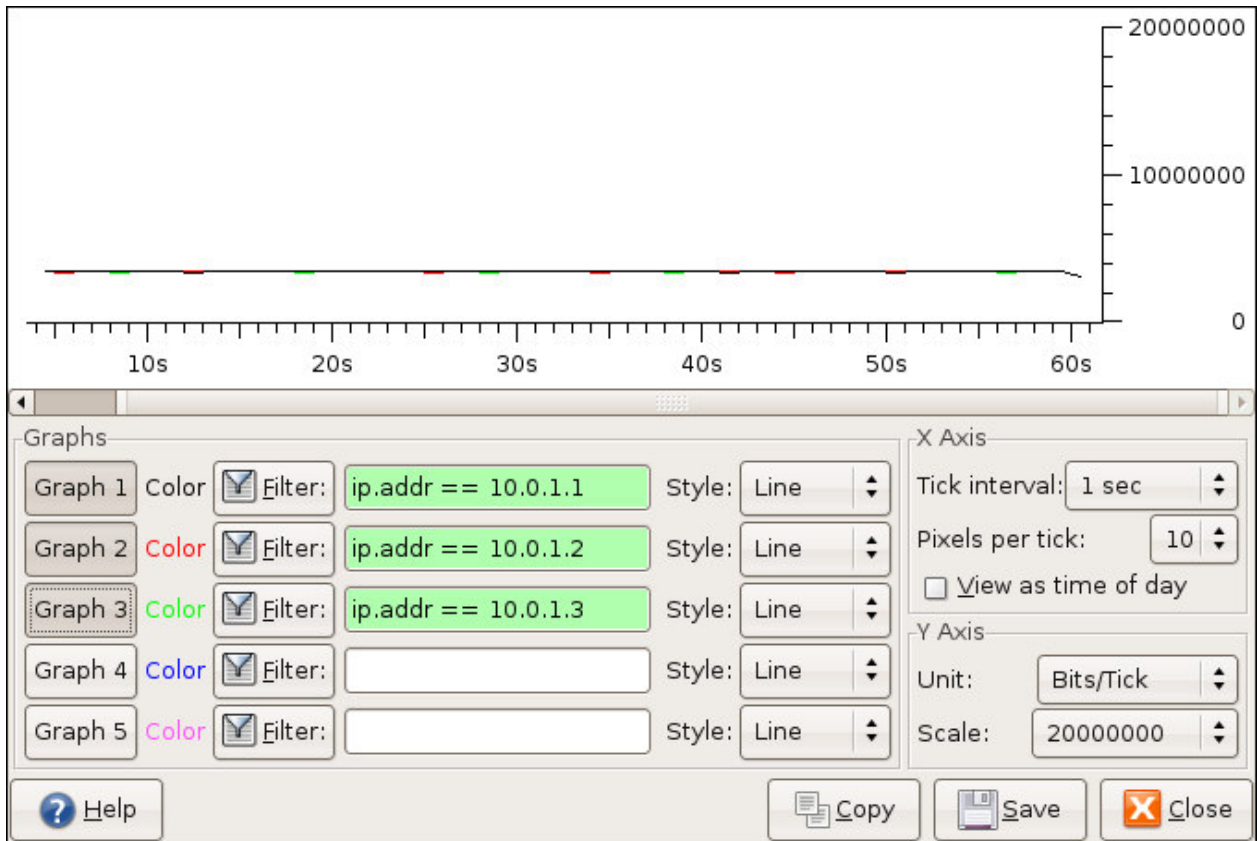


Figure 6

## 4 Max-reserved-bandwidth

### 4.1 The reason for this document

Now we can take it up a notch and test the max-reserved-bandwidth command. Cisco states the following:

To change the percent of interface bandwidth allocated for Resource Reservation Protocol (RSVP), class-based weighted fair queueing (CBWFQ), low latency queueing (LLQ), IP RTP Priority, Frame Relay IP RTP Priority, and Frame Relay PVC Interface Priority Queueing (PIPQ), use the max-reserved bandwidth command in interface configuration mode. To restore the default value, use the no form of this command.

So according to Cisco's logic, if for example max-reserved-bandwidth on an interface is 75, only 75 percent could be reserved for CBWFQ in our case, since CBWFQ is all that we are using. Default value could be 75, but we better not rely on it and always put in what we want to experiment with.

We will make the test as simple as possible. R1, R2 and R3 will be generating traffic, but only R2 will receive any reservation whatsoever. We will configure small reservation of 5 percent from the total interface bandwidth.

After several minutes of measuring, we check the output.

```
R4#sh policy-map int fa 0/1
FastEthernet0/1

Service-policy output: T4

Class-map: R2 (match-all)
 276978 packets, 419344692 bytes
 30 second offered rate 9843000 bps, drop rate 9350000 bps
Match: access-group 2
Queueing
queue limit 100 packets
(queue depth/total drops/no-buffer drops) 100/263130/0
(pkts output/bytes output) 13849/20967386
bandwidth 5% (500 kbps)

Class-map: class-default (match-any)
 553995 packets, 838693224 bytes
 30 second offered rate 19686000 bps, drop rate 10336000 bps
Match: any

queue limit 100 packets
(queue depth/total drops/no-buffer drops) 100/290830/0
(pkts output/bytes output) 263167/398381940
```

It seems that really R2 received 5% bandwidth while R1 and R3 received the rest. Let's check Wireshark.

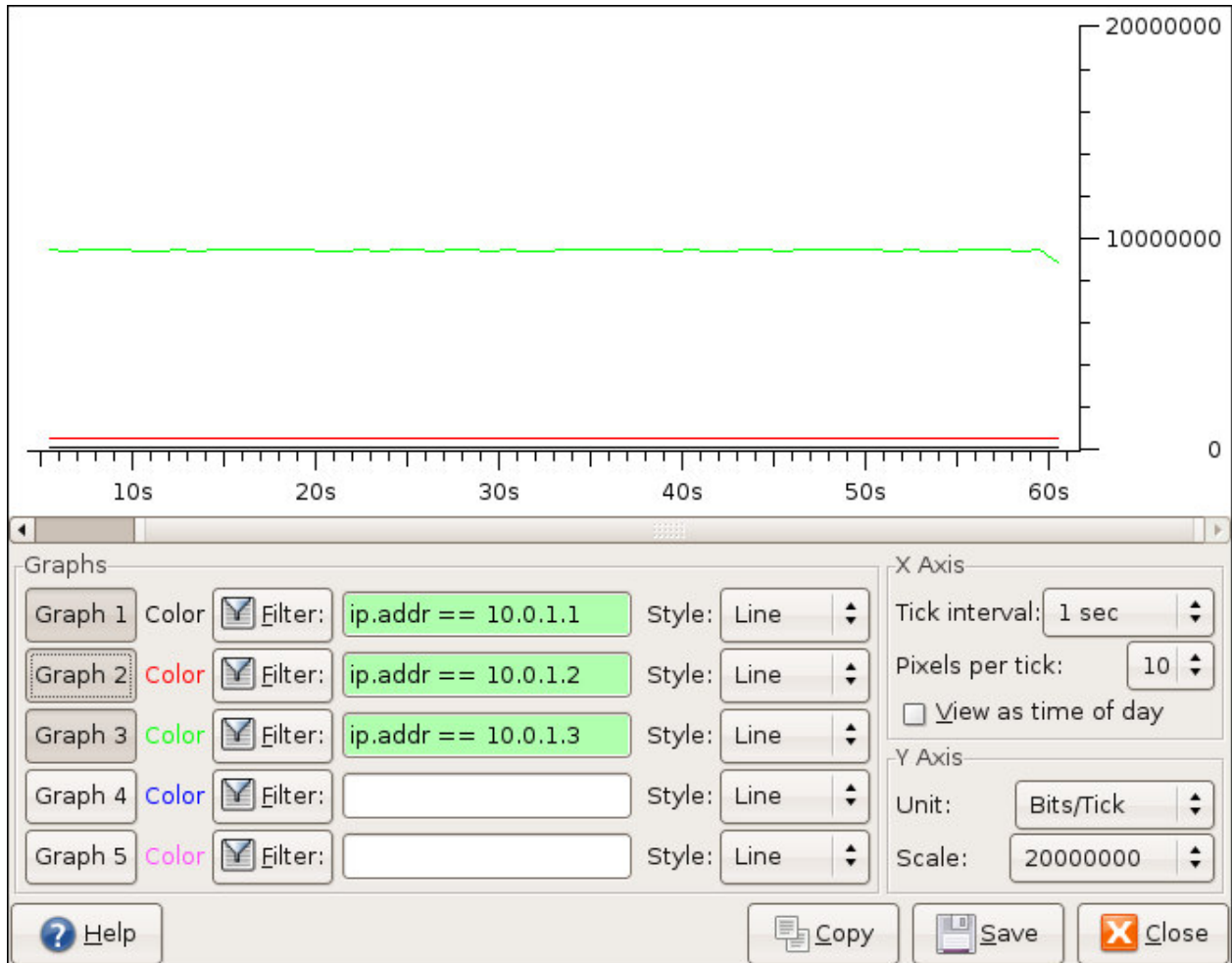


Figure 7

But what's this? R3 receiving all the glory, R2 getting it's 500Kbps, but what about R1?  
From the graph it appears that almost all of it's packets are lost, an enormous drop rate.

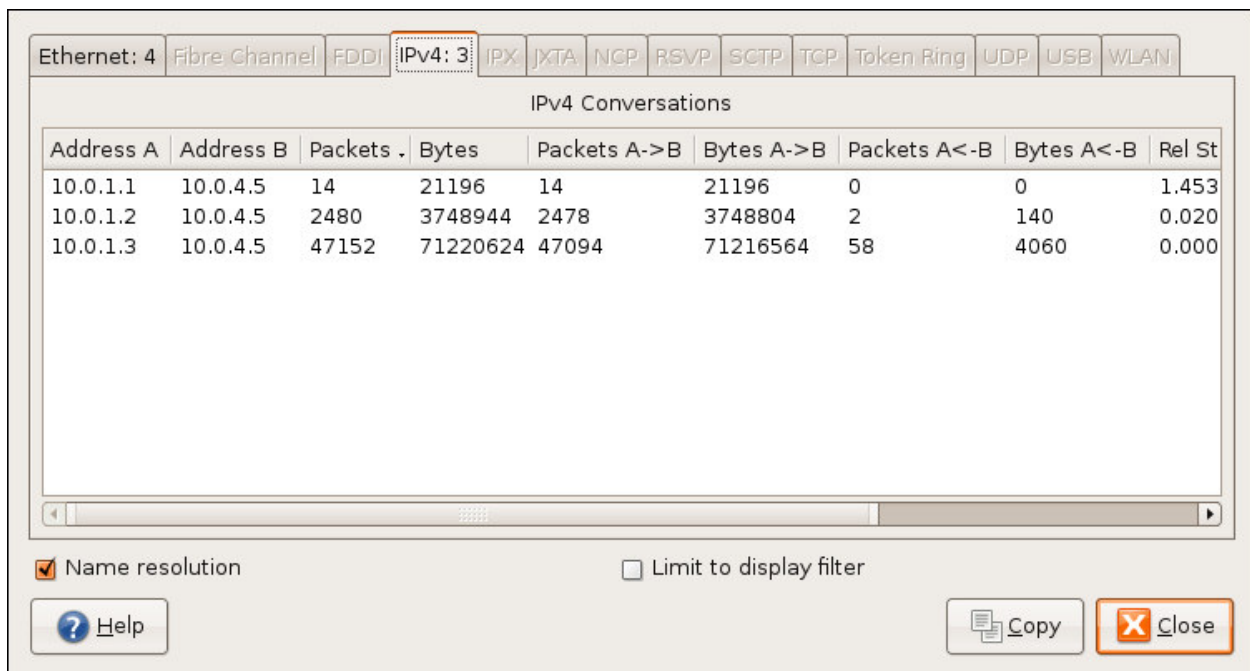


Figure 8

Wireshark's analysis of conversations confirms that only 14 packets from R1 were transmitted. Previously we saw that R4 reported a drop rate of 10336000 bps on the output of Fa 0/1. Just to be sure, let's check input counters on R4.

```
R4#show policy-map interface fa 0/0
FastEthernet0/0

Service-policy input: INPUT-MONITOR

Class-map: R1 (match-all)
 146589 packets, 221935746 bytes
 30 second offered rate 9826000 bps
 Match: access-group 1

Class-map: R2 (match-all)
 146588 packets, 221934232 bytes
 30 second offered rate 9826000 bps
 Match: access-group 2

Class-map: R3 (match-all)
 146588 packets, 221934232 bytes
 30 second offered rate 9826000 bps
 Match: access-group 3

Class-map: class-default (match-any)
 0 packets, 0 bytes
 30 second offered rate 0 bps, drop rate 0 bps
 Match: any
```

This is very bad behavior by CBWFQ. While all three routers sent the same amount of data, IOS chooses not to transmit data from R1. The only difference in the packets, really, is the source IP address. This could be the problem with underlying WFQ, classifying R1's traffic somewhere else than R3's traffic. To correct this, we will use "fair-queue" in class-default. Fair-queue enables WFQ for the class and the only class which supports WFQ is class-default. Only in the newest IOS 12.4.20T (and newer) it is possible to use Fair-Queue on any class.

To correct this, a new policy-map is created and new measurement is made.

```
policy-map T5
class R2
  bandwidth percent 5
  queue-limit 100 packets
class class-default
  queue-limit 100 packets
  fair-queue
```

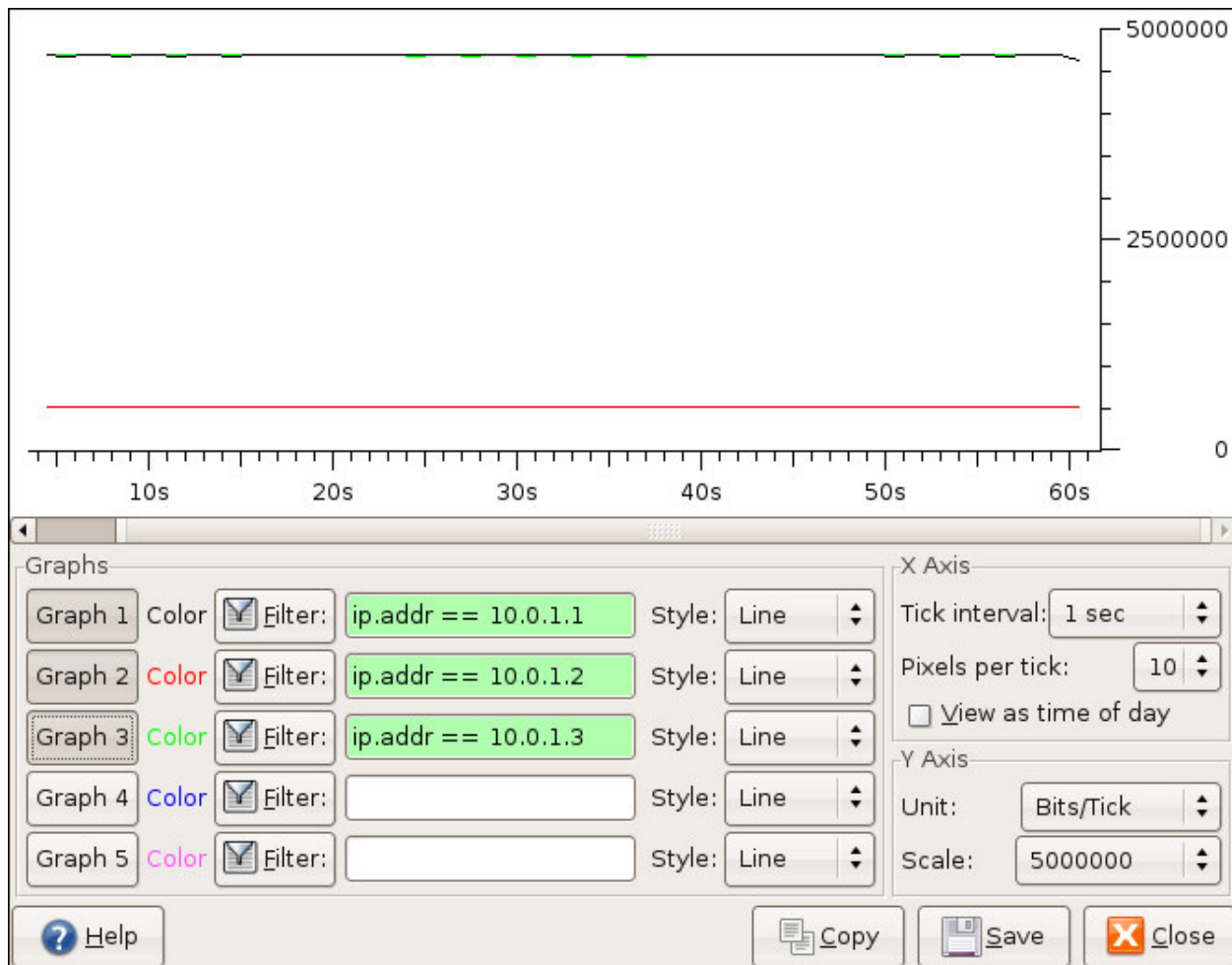


Figure 9

Now R2 is getting 500 Kbps, while R1 and R3 are getting the rest. But should they really? On the interface we have the max-reserved-bandwidth set to 75, so in theory R1 and R3 together should get 25% of interface bandwidth, while R2 should receive all reservable bandwidth. Remember, according to Cisco the command is there to “change the percent of interface bandwidth allocated for Resource Reservation Protocol (RSVP), class-based weighted fair queueing (CBWFQ)...”

Let's up that reservation to 100 % and repeat the experiment.

#### R4 configuration

```
interface FastEthernet0/1
max-reserved-bandwidth 100
service-policy output T5
```

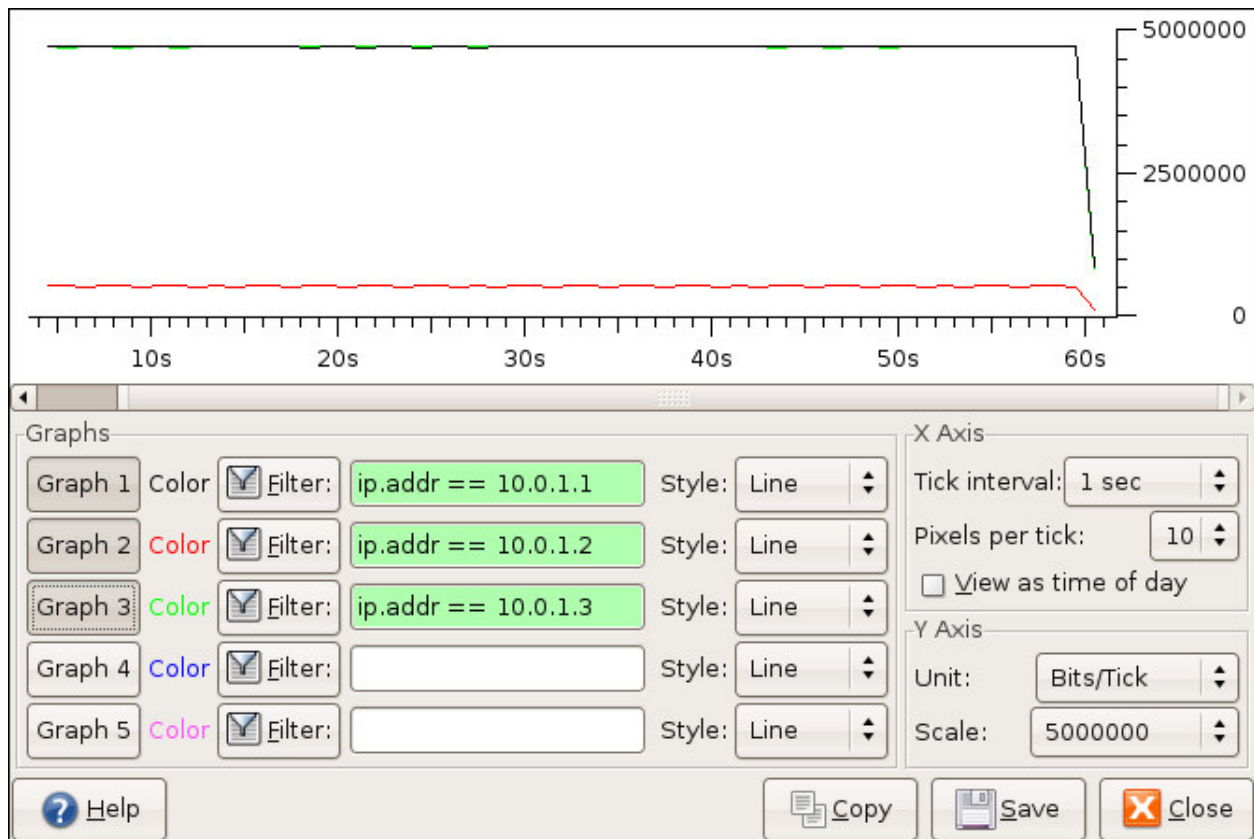


Figure 10

Theoretically, since we allowed CBWFQ to reserve 100%, we should have starved class-default with R2 traffic. This did not happen, and in fact, there was no change whatsoever. Finally, let's try lowering the value to 10% and repeat the experiment.

#### R4 configuration

```
interface FastEthernet0/1
max-reserved-bandwidth 10
service-policy output T5
```

### This resulted in the following

```
R4#sh policy-map int fa 0/1
FastEthernet0/1

Service-policy output: T5

Class-map: R2 (match-all)
 423576 packets, 641294064 bytes
 30 second offered rate 9843000 bps, drop rate 9351000 bps
 Match: access-group 2
 Queueing
  queue limit 100 packets
 (queue depth/total drops/no-buffer drops) 100/402395/0
 (pkts output/bytes output) 21182/32069548
 bandwidth 5% (500 kbps)

Class-map: class-default (match-any)
 847213 packets, 1282593051 bytes
 30 second offered rate 19686000 bps, drop rate 10335000 bps
 Match: any
 Queueing
  queue limit 100 packets
 (queue depth/total drops/no-buffer drops/flowdrops) 52/444759/0/444759
 (pkts output/bytes output) 402455/609229439
 Fair-queue: per-flow queue limit 25
```

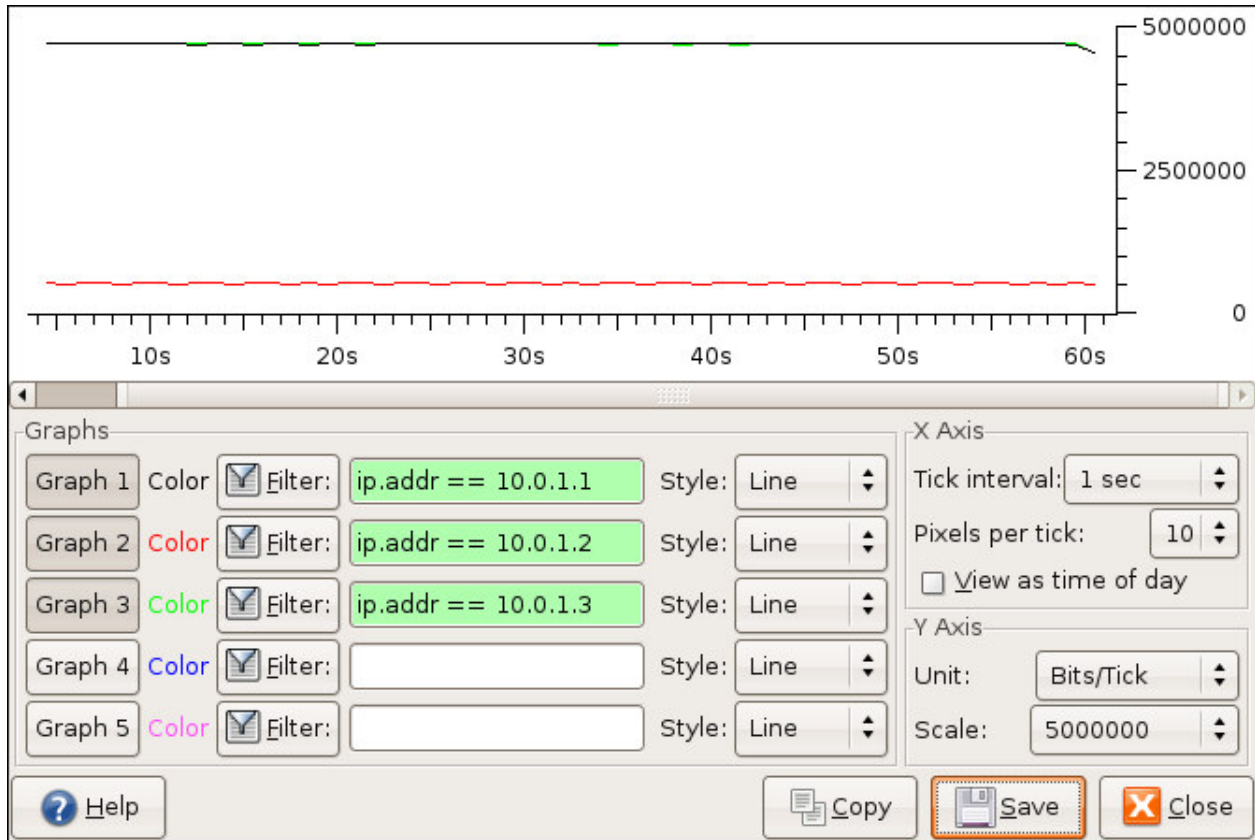


Figure 11

Again, the results are almost identical. As if the developers were expecting this test, the 12.4.20T1 reported a log message, a comment of some sort, after I have changed “max-reserved-bandwidth” command:

```
Dec 7 00:11:36.162: Interface FastEthernet0/1 max_reserved_bandwidth config will not take effect on the queuing features configured via service-policy
```

There is no facility or level reported in this message. Only date and time, and the message body. That is why I added “Almost” in the introduction. Cisco lost the bet documentation wise, but IOS tries to get things straight by almost talking to me.

## 5 Conclusion

### 5.1

Max-reserved-bandwidth command does not have any effect on queuing system whatsoever. The newer IOS's do not even check policy-maps total reservation parameters against max-reserved-bandwidth settings. The newest IOS's even notify you of this fact, as if to change the documentation error.

## 6 PRE 12.4.20T

### 6.1 CBWFQ mechanism and class-default starvation.

Before 12.4.20T, and now I am not sure in exactly which release the situation has changed, there was a possibility of starvation for every class that does not have at least one bit guaranteed using either “bandwidth” or “priority” commands. The situation was ridiculous in a sense that if we reserve a tiny amount for at least one class, all the traffic in classes without reservation, be it class-default or not, would simply starve.

This was largely due to nuances of CBWFQ implementation. To demonstrate this, I got myself a good old 12.4(2)T6 and put it on R4. I left configuration unchanged and created a new policy-map for experimentation.

```
policy-map T7
class R1
  bandwidth percent 10
service-policy output T7
```

Then applied it on the output interface of R4

```
interface FastEthernet0/1
ip address 10.0.4.4 255.255.255.0
load-interval 30
duplex full
speed 10
service-policy output T7
```

After a while, I checked the output

```
R4#sh policy-map int fa 0/1
FastEthernet0/1

Service-policy output: T7

Class-map: R1 (match-all)
 583970 packets, 884130580 bytes
 30 second offered rate 9843000 bps, drop rate 94000 bps
Match: access-group 1
Queueing
  Output Queue: Conversation 265
  Bandwidth 10 (%)
  Bandwidth 1000 (kbps) Max Threshold 64 (packets)
  (pkts matched/bytes matched) 583970/884130580
  (depth/total drops/no-buffer drops) 64/5529/0

Class-map: class-default (match-any)
 1168021 packets, 1768263701 bytes
 30 second offered rate 19688000 bps, drop rate 19594000 bps
Match: any
```

And confirmed the findings with Wireshark. Traffic from R2 and R3 is almost nonexistent.

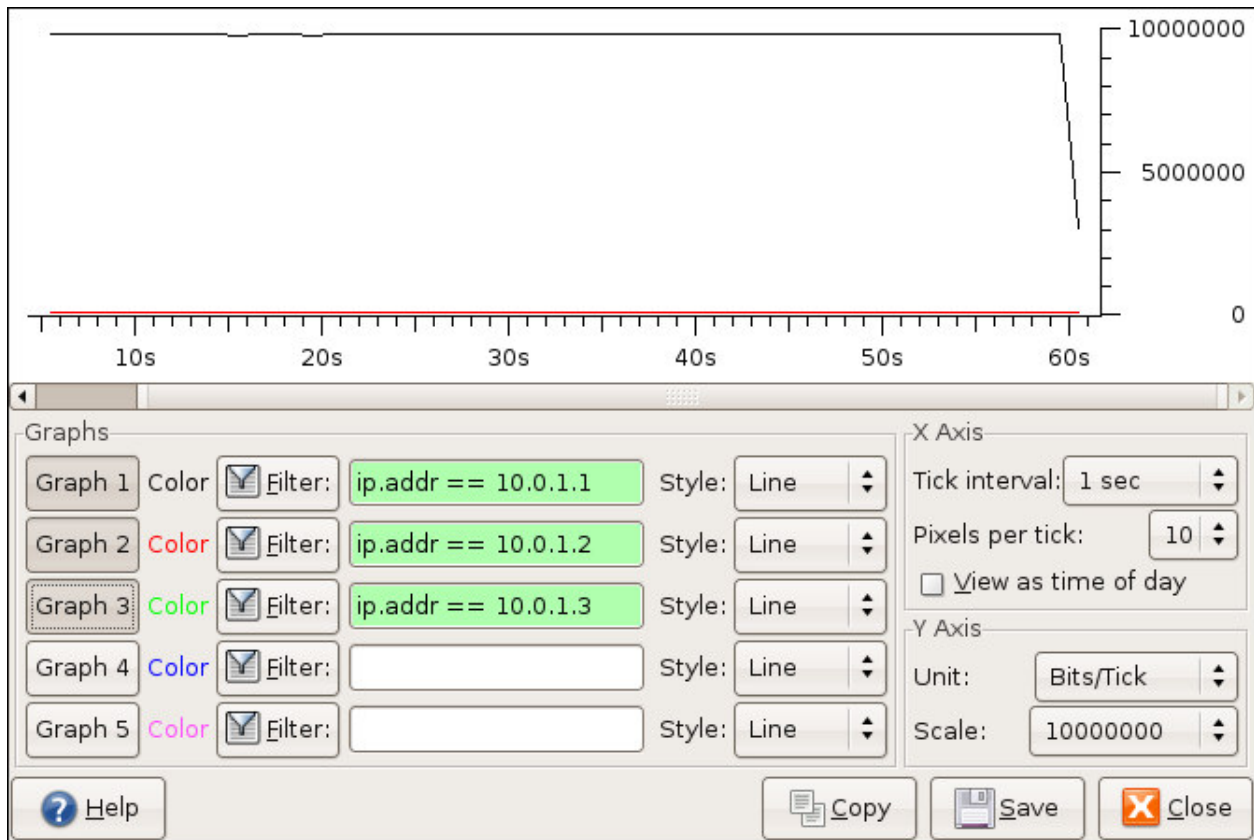


Figure 12

The figure shows the 'IPv4 Conversations' window in Wireshark. The 'IPv4: 3' tab is selected. The table below shows the data:

Address A	Address B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B	Rel St
10.0.1.3	10.0.4.5	231	349734	231	349734	0	0	0.504
10.0.1.2	10.0.4.5	232	351248	232	351248	0	0	0.244
10.0.1.1	10.0.4.5	48672	73516128	48612	73511928	60	4200	0.000

Additional settings include: Name resolution: checked, Limit to display filter: unchecked. Buttons for Help, Copy, and Close are visible at the bottom.

Figure 13

How can CBWFQ create such a mess? The answer lies in the unpublished algorithm. What it does is basically create a WFQ queue for every class and then assign same weight to the packets in that class. The weight was derived from inverse value of percentual reservation. Meaning if you used absolute bandwidth, bandwidth percent or bandwidth remaining percent, all the values were converted to “percent” of interface bandwidth. Then a “magic number” of 1551 or 3096 (depending on IOS) was divided by that number, resulting in weight. So in this case, the reservation was 10 percent. In this IOS magic number is 1551. So to get class weight, CBWFQ divided 1551 by 10, and got 155. To check this, we simply issue an old but very very useful command “show queue”. This command shows us the actual queues used by the queueing system at the very instant of issuing the command.

Here is the output:

```
R4#sh queue fa 0/1
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 11994
  Queueing strategy: Class-based queueing
  Output queue: 128/1000/64/11994 (size/max total/threshold/drops)
    Conversations 3/4/256 (active/max active/max total)
    Reserved Conversations 1/1 (allocated/max allocated)
    Available Bandwidth 6500 kilobits/sec

  (depth/weight/total drops/no-buffer drops/interleaves) 63/155/46/0/0
  Conversation 265, linktype: ip, length: 1514
  source: 10.0.1.1, destination: 10.0.4.5, id: 0xC001, ttl: 254, prot: 1

  (depth/weight/total drops/no-buffer drops/interleaves) 32/32384/5976/0/0
  Conversation 34, linktype: ip, length: 1514
  source: 10.0.1.3, destination: 10.0.4.5, id: 0xE1F2, ttl: 254, prot: 1

  (depth/weight/total drops/no-buffer drops/interleaves) 33/32384/5974/0/0
  Conversation 33, linktype: ip, length: 1514
  source: 10.0.1.2, destination: 10.0.4.5, id: 0xAF4C, ttl: 254, prot: 1
```

CBWFQ always looks at Sequence Numbers (or Finish Time) and sends the packet with the lowest value. Now, if we use the standard WFQ formula:

$$\text{SN}(\text{of new packet}) = \text{SN}(\text{of previous packet in the flow}) + \text{Size}(\text{current packet}) * 32384 / (\text{IPP} + 1)$$

We can see that the other two flows (10.0.1.3 and 10.0.1.2 in the output) will receive very bad service. If you'll plug these values into Excel or other spreadsheet processor you'll see that the result of the wireshark sniffer corresponds with the formula.

## 7 Post 12.4.20T CBWFQ Notes

### 7.1 Reservation types

Reservation types cannot be mixed at all. Before, we could reserve percent or absolute, and use “Remaining Percent” in other classes. Now the IOS always reports the following:

```
Mixed bandwidth types are not supported. Pls configure bandwidth command either in kbps, percent, remaining percent or remaining ratio but not mixed
```

### 7.2 Bandwidth Remaining Percent

Because “max-reserved-bandwidth” command has no effect whatsoever, the decided to stop even calculating “bandwidth remaining percent” based on it. Now “bandwidth percent” and “bandwidth remaining percent” are identical if “priority” is not used.

```
policy-map E1
class R1
  bandwidth remaining percent 50
class R2
  bandwidth remaining percent 49

R4(config)#int fa 0/1
R4(config-if)#max-reserved-bandwidth 30
Reservable bandwidth is being reduced.
Some existing reservations may be terminated.
R4(config-if)#
R4#sh policy-map int fa 0/1 | in bandwidth|Service
Service-policy output: E1
  bandwidth remaining 50% (5000 kbps)
  bandwidth remaining 49% (4900 kbps)

R4(config)#int fa 0/1
R4(config-if)#max-reserved-bandwidth 90
R4#sh policy-map int fa 0/1 | in bandwidth|Service
Service-policy output: E1
  bandwidth remaining 50% (5000 kbps)
  bandwidth remaining 49% (4900 kbps)
```

### 7.3 Show Queue

Unfortunately, the “show queue” command that I previously extensively used to troubleshoot and discover operation of CBWFQ and LLQ is not available anymore. IOS tells me to basically go away

```
R4#show queue fa 0/1
show queue command is deprecated. Please use show policy-map interface
```

## 7.4 Debug QoS

Interesting new debug qos commands have appeared, good use of which I have yet to discover.