

CESNET Technical Report 3/2002

MDRR on Cisco GSR with Gigabit Ethernet

Sven Ubik, ubik@cesnet.cz

April 15, 2002

1 Experiment objective

Verify functionality of MDRR and policing on a 3-port Gigabit Ethernet line card on Cisco GSR12008. Testing of WRED was not our objective, but we used it in order to enable MDRR as explained below.

2 Test configuration

- Each PC had a Gigabit Ethernet adapter.
- A testing stream generated by PC1 was sent as input to port 1.

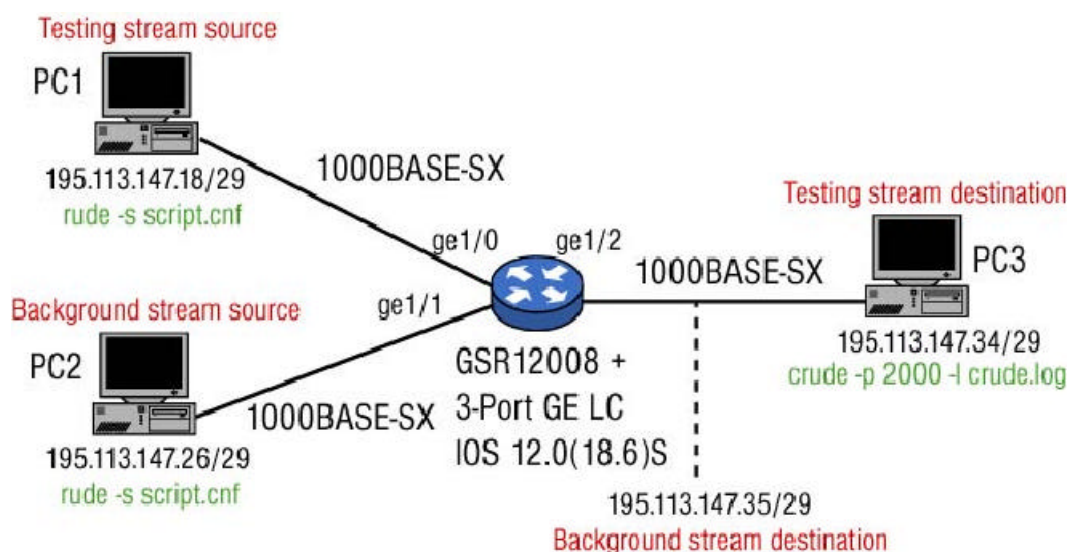


Figure 1: Test configuration

- A background stream generated by PC2 was sent as input to port 2.
- The testing stream was destined to PC3 connected to port 3. The background stream was destined to a MAC address manually set to be present on port 3 (no real PC was at that MAC address). That is, port 3 was shared by both streams.
- Unless noted otherwise, both streams consisted of 1500-byte packets.
- Rate is indicated for each test.

Note about the IOS version:

We first used IOS 12.0(18.5)ST. We could not ping from the GSR to the PCs. Debug revealed that the GSR sent an ARP request, the PC sent an ARP reply, but the GSR did not receive this ARP reply for unknown reason. After several days of debugging we finally found that changing IOS to 12.0(8)S (that is from one early-deployment release to another early-deployment release, because nothing else than early-deployment was available for a \$120000 box) rectified the problem.

3 Back-to-back configuration

We directly connected PC1 and PC3 (back-to-back) to check the maximum achievable throughput and one-way delay without any router present. The packet generator on PC1 was set to produce a stream with bandwidth increasing from 50 Mb/s to 1 Gb/s in 50 Mb/s steps.

Throughput and delay for 1500-byte packets is shown in Fig. 2. Throughput and delay for 128-byte packets is shown in Fig. 3. Maximum throughput for 1500-byte packets was 542 Mb/s and maximum throughput for 128-byte packets was 60 Mb/s. However, this throughput was limited mainly by the receiving PC. That is the sending PC was able to produce higher bandwidth, but the receiving PC was not able to process that bandwidth. According to messages printed by RUDE packet generator, the sending PC was able to produce about 850 Mb/s in 1500-byte packets. We used this capability for a background stream which had to be only produced, but not received. Delay is indicated for that part of the test when the receiving PC was still able to receive the stream.

Maximum delay for 1500-byte packets was 166 us, while average delay was 55 us. Maximum delay for 128-byte packets was 350 us, while average delay was 26 us.

To check if PC1 and PC3 were able to steadily process bandwidth within the acceptable range, we set the packet generator to produce a constant 250 Mb/s stream of 1500-byte packets and a constant 25 Mb/s stream of 128-byte packets for 30 seconds. Throughput and delay for 1500-byte packets is shown in Fig. 4. Throughput and delay for 128-byte packets is shown in Fig. 5.

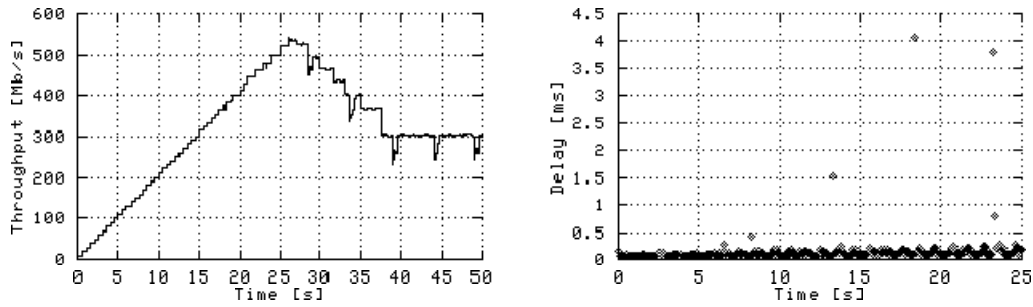


Figure 2: Directly connected, 1500-byte packets

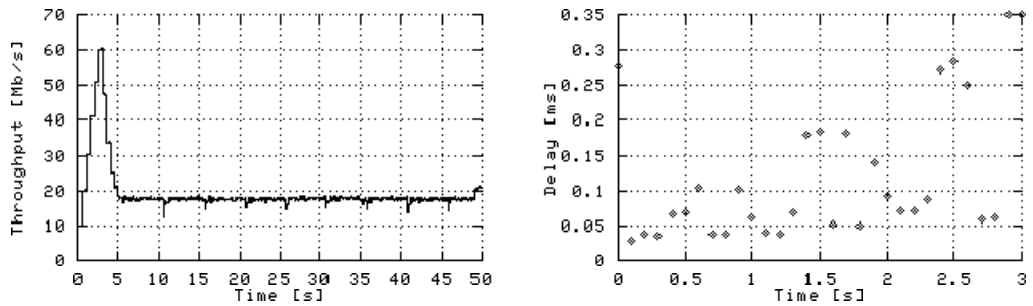


Figure 3: Directly connected, 128-byte packets

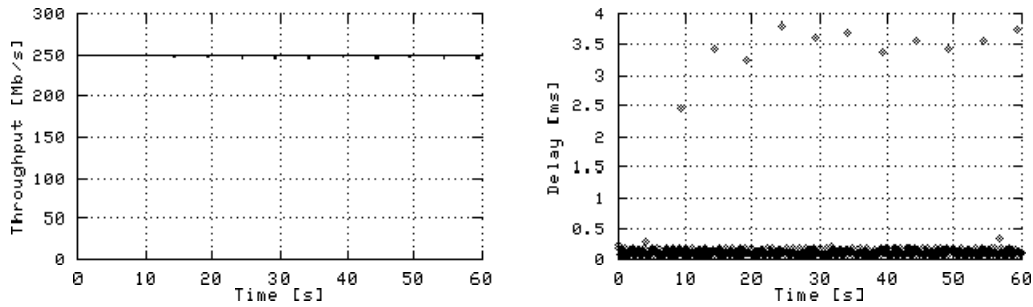


Figure 4: Directly connected, 1500-byte packets

It is clear that the set throughput was steadily achieved. However, about every 5 seconds there was a short peak in delay. In case of 1500-byte packets, there were also short drops in bandwidth. Each drop represented about 32 lost packets. We do not know why this happened, but we must take this effect into consideration when observing streams that passed through a router.

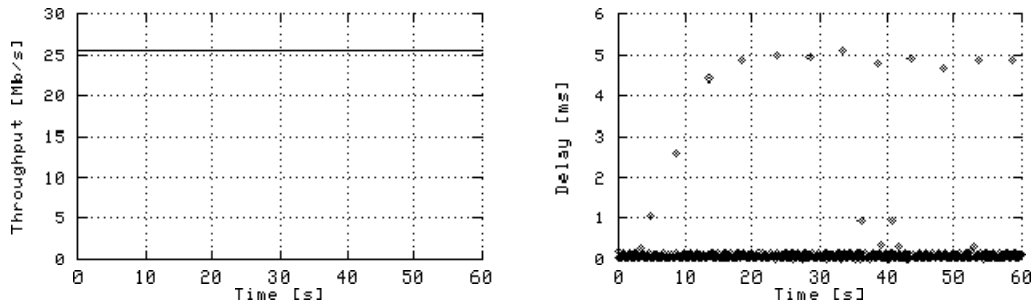


Figure 5: Directly connected, 128-byte packets

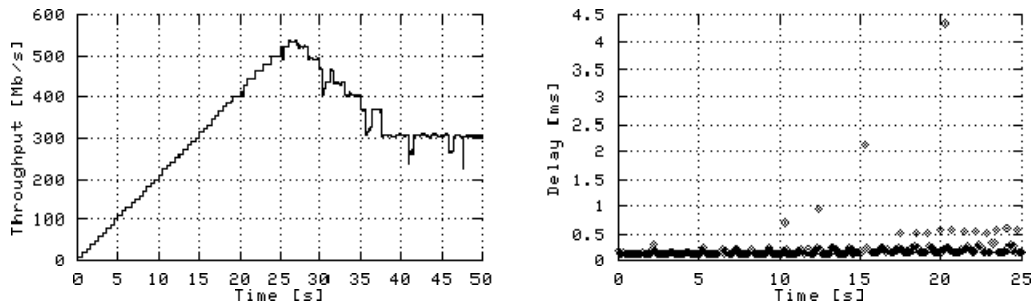


Figure 6: GSR, 1500-byte packets

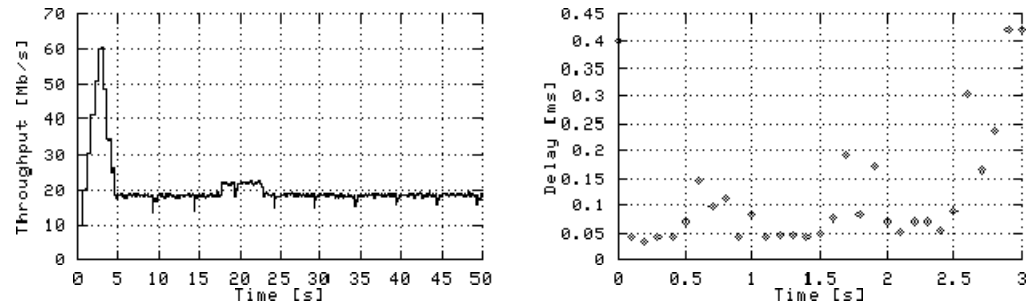


Figure 7: GSR, 128-byte packets

4 Inserting a GSR

We inserted a GSR between PC1 and PC3 to see its influence on QoS characteristics. No special traffic handling was applied (no MDRR or WRED).

Throughput and delay for 1500-byte packets is shown in Fig. 6. Throughput and delay for 128-byte packets is shown in Fig. 7. There is no visible difference in throughput or delay whether there was a GSR between the PCs or a direct wire.

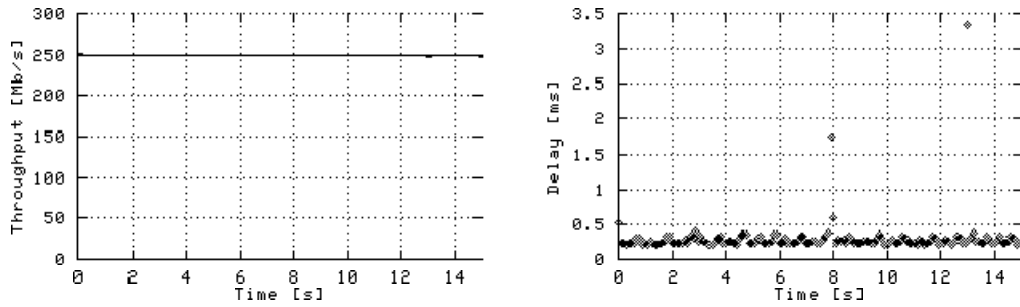


Figure 8: GSR, 250 Mb/s testing stream + 700 Mb/s background stream

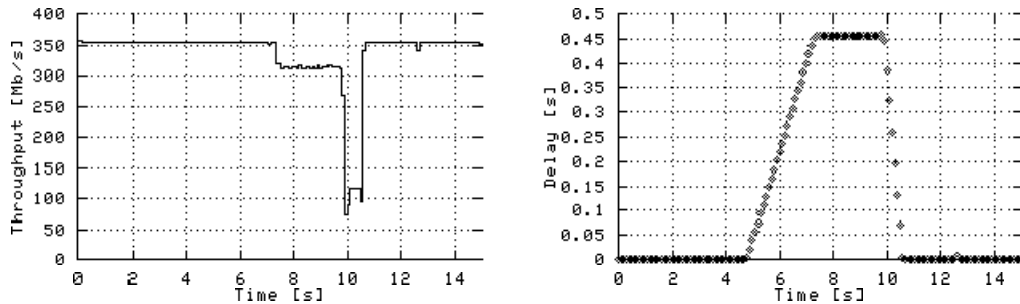


Figure 9: GSR, 350 Mb/s testing stream + 800 Mb/s background stream

5 Applying a background stream

To make a test at a higher bandwidth, we connected PC2 to another port on GSR and sent a background stream from this PC. The testing stream was generated for 15 seconds. The background stream was added in period from 5 to 10 seconds.

Throughput and delay of 250 Mb/s testing stream with 700 Mb/s background stream (total of 950 Mb/s) is shown in Fig. 8. Throughput and delay of 350 Mb/s testing stream with 800 Mb/s background stream (total of 1150 Mb/s) is shown in Fig. 9. At 950 Mb/s there was no visible difference in throughput or delay whether the background stream was applied or not. At 1150 Mb/s there was obviously a drop in throughput when the background stream is applied. For more than two seconds (!) the GSR buffered incoming traffic resulting in 0.45 second (!) delay before the buffer was full. Because we were sending 1150 Mb/s in the GSR and we could send only 1000 Mb/s out of the GSR, there was a difference of 150 Mb/s that was accumulated in a buffer. After two seconds it totaled 300 Mb, that is 37.5 MB. We did not find a possibility to configure this buffer size. When the background stream was switched off at 10 seconds, there was a big drop in throughput of the testing stream to about 100 Mb/s for about 0.7 second. The reason of this drop is not clear.

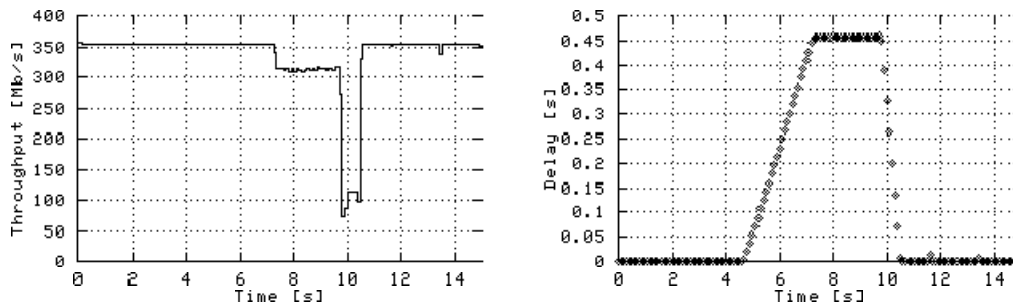


Figure 10: GSR, 350 Mb/s testing stream + 800 Mb/s background stream, MDRR (no WRED)

6 MDRR + WRED

We sent the testing stream with IP priority set to 1 (TOS byte=0x20) and the background stream with IP priority set to 0. We tried to reserve 35 and 65 following configuration (MDRR only, no WRED):

```
interface GigabitEthernet0/2
 ip address 195.113.147.33 255.255.255.248
 no negotiation auto
 tx-cos group1
 !
 cos-queue-group group1
 precedence 1 queue 1
 queue 0 65
 queue 1 35
```

Throughput and delay of the testing stream is shown in Fig. 10. Throughput dropped significantly when the background stream was applied.

One possible reason why MDRR did not work could be contention in "tofab" queues (before the switching engine). We checked the state of both "tofab" and "frfab" (after the switching engine) queues when both the testing and background streams were running for one minute. The state of "tofab" queues did not change by applying the packet streams. On the other hand, "frfab" queues corresponding to the size of packets were depleted. So there was not contention before the switching engine, only after the switching engine. The output of the monitoring commands executed on the line card was as follows (edited for clarity):

```
LC-Slot0>sh controllers tofab queues
Qnum Head Tail #Qelem LenThresh
```

3 non-IPC free queues:

126116/126116 (buffers specified/carved), 51.97%, 80 byte data size
1 89005 89004 126116 262143

77609/77609 (buffers specified/carved), 31.98%, 608 byte data size
2 197670 197669 77609 262143

38804/38804 (buffers specified/carved), 15.99%, 1568 byte data size
3 228338 228337 38804 262143

LC-Slot0>sh controllers frfab queues

Qnum Head Tail #Qelem LenThresh

3 non-IPC free queues:

118419/118419 (buffers specified/carved), 51.97%, 80 byte data size
1 896 895 118419 262143

72872/72872 (buffers specified/carved), 31.98%, 608 byte data size
2 139751 177047 37294 262143

36436/36436 (buffers specified/carved), 15.99%, 1568 byte data size
3 192373 192373 1 262143

We tried to add WRED configuration as follows:

```
cos-queue-group group1
precedence 0 random-detect-label 0
precedence 1 random-detect-label 1
random-detect-label 0 1000 2000 1
random-detect-label 1 2000 3000 1
```

Throughput and delay of the testing stream is shown in Fig. 11. The full throughput was maintained when the background stream was applied, with the exception of a short drop in throughput at the point when the background stream was switched off. What was the reason of this drop? However, maintenance of throughput in this test was owing to dropping packets by WRED, rather than bandwidth sharing by MDRR. Because we configured WRED so that the dropping probability of 1 was reached for the background stream just at the beginning of the dropping range for the testing stream (when its dropping probability was still 0), as much traffic of the background stream was dropped as needed to forward all testing stream traffic.

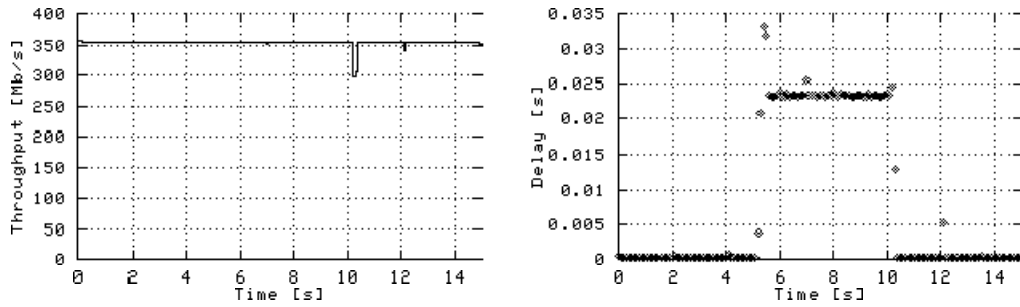


Figure 11: GSR, 350 Mb/s testing stream + 800 Mb/s background stream, MDRR + WRED

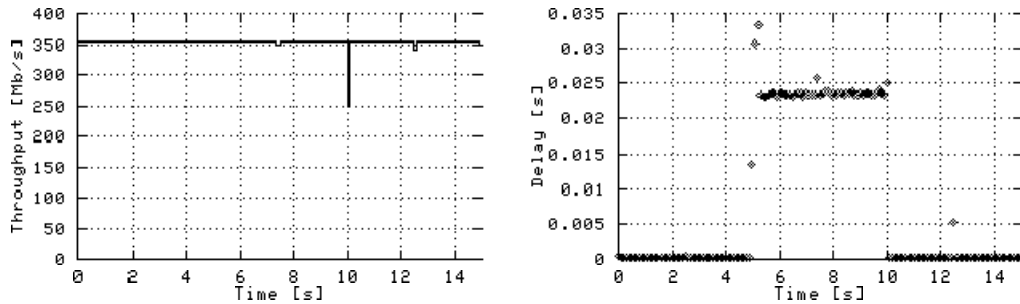


Figure 12: GSR, 350 Mb/s testing stream + 800 Mb/s background stream, MDRR (30%+70%) + WRED

To verify this effect, we decreased the bandwidth share for the testing stream to 30

```
cos-queue-group group1
precedence 1 queue 1
queue 0 70
queue 1 30
```

Throughput and delay of the testing stream is shown in Fig. 12. The full throughput was maintained when the background stream was applied even though lower share of bandwidth was dedicated for the testing stream.

We tried to set the same dropping range and probability for both the testing stream and the background stream as follows (we could also use one random-detect-label command for both precedence values):

```
cos-queue-group group1
precedence 0 random-detect-label 0
precedence 1 random-detect-label 1
random-detect-label 0 1000 2000 1
random-detect-label 1 1000 2000 1
```

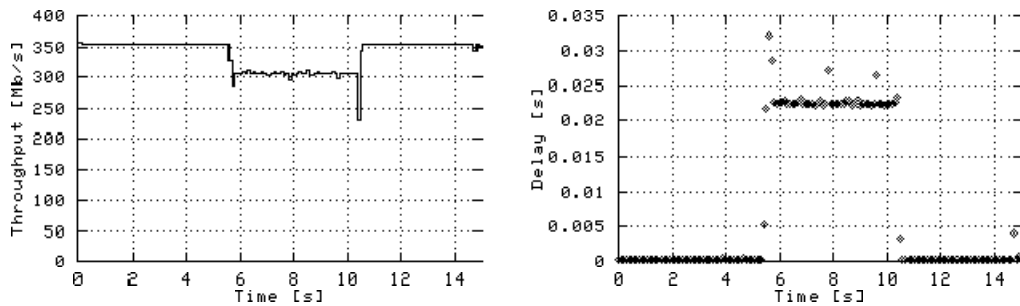


Figure 13: GSR, 350 Mb/s testing stream + 800 Mb/s background stream, MDRR (30%+70%) + WRED (same dropping range)

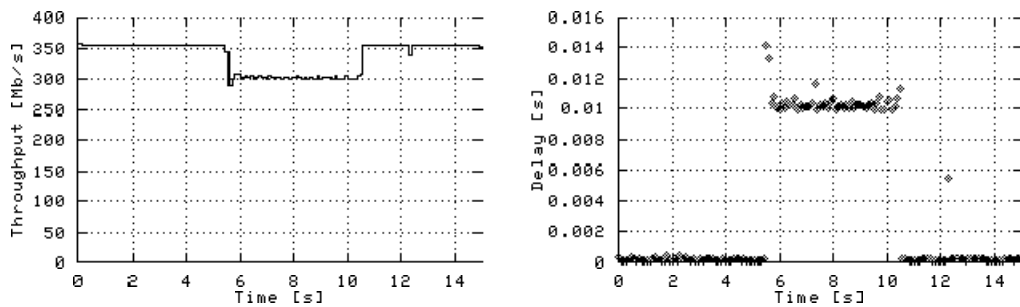


Figure 14: GSR, 350 Mb/s testing stream + 800 Mb/s background stream, MDRR (30%+70%) + WRED (lower dropping range)

Throughput and delay of the testing stream is shown in Fig. 13. MDRR now divide the link bandwidth between each stream according to shares assigned to individual queues. Each queue filled up to the point where its dropping probability discarded just as many packets as needed to balance the difference between incoming rate of packets with corresponding IP priority and the bandwidth share assigned to that queue.

However, our impression is that MDRR should work also without WRED. The purpose of MDRR and WRED is completely different. The task of MDRR is to divide the available bandwidth according to specified shares, whereas the task of WRED is to prevent congestion and provide optimal link bandwidth utilization using the TCP congestion control mechanism. Nevertheless, it seems that MDRR cannot be used without WRED on Cisco GSR.

We tried to use lower limits of WRED dropping range as follows:

```

ip<
  ipre< cos-queue-group group1 precedence 0 random-detect-label 0 precedence 1
  random-detect-label 1 random-detect-label 0 100 200 1 random-detect-label 1 100
  200 1 ;/pre<

```

Throughput and delay of the testing stream is shown in Fig. 14. Lower limits of the dropping range meant earlier dropping, shorter queues and lower delay.

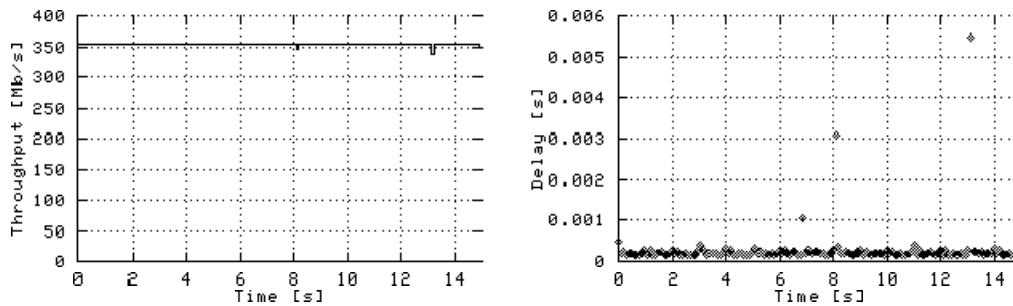


Figure 15: GSR, 350 Mb/s testing stream + 800 Mb/s background stream, strict priority

7 Priority

Finally, we tried using a strict priority instead of MDRR and WRED (configuration to be added here). Throughput and delay of the testing stream is shown in Fig. 15. Strict priority provided not only lower delay than bandwidth sharing using MDRR and WRED, but also lower loss rate. Looking at Fig. 13, the full throughput was maintained. However, precise measurement of losses revealed 0.22only 0.04

8 Conclusion

- MDRR worked very well to protect the testing stream, but only when WRED was also enabled.
- When MDRR was not used and the line was congested, the testing stream experienced significant short drop in throughput when the background stream was switched off.
- Average delay was slightly increasing when the line was significantly loaded, but not congested. Could be an error of our measurement method.