Simulating quality of service

UNIK4200

Joachim Tingvold

University of Oslo

Bergen University College

1

Introduction

To get a better understanding of how QoS works, we can simulate it. There are several tools one can use to achieve this, but we'll use J-Sim[1], a java-based simulation tool. Our task is to run a few simulations, wrapped in a TCL-script, and interpret the results. What do we see? Why is it happening? Is it reasonable?

1.1 Topology

We use the same network topology for all three tests. The topology used for all the simulations, consists of 3 routers (1 core, 2 edge), and 8 hosts. A drawing of the topology, can be seen on Figure 1.1.

[1]http://sites.google.com/site/jsimofficial/

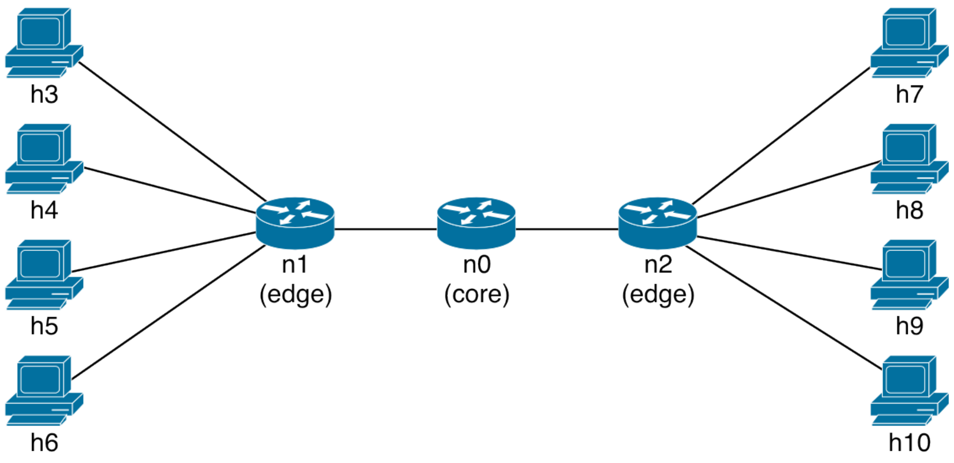

Figure 1.1: Network topology - Overview of the network topology.

2

Simulation

We have eight hosts, paired into four pairs. Three of these pairs, will transmit TCP, whilst the last pair, will transmit UDP. The TCP-pairs is as follows; h3 $\rightarrow$ h7, h4 $\rightarrow$ h8, h6 $\rightarrow$ h10. And the UDP-pair; h5 $\rightarrow$ h9. All TCP-hosts gets their MSS (Maximum Segment Size) and packet-size set to 512 byte. The time-duration for all three simulations, is 200 seconds.

2.1 Without QoS

This simulation has no QoS. The links between the hosts/routers, are 10Mbps. The links to/from router n0, however, is set to 1Mbps. In addition, interface 0 on router n0, is given a limited buffer size of 20.

The UDP-stream from h5 to h9, and the TCP-streams from h3 to h7 and h4 to h8, is started at the very beginning. After 50 seconds, the TCP-stream from h6 to h10 is also started.

After running the simulation, we can see from Figure 2.1 that the UDP-stream is slightly affected by the congestions that occur (even though the changes are really small). We also see that the throughput changes according to the congestion window (as seen on Figure 2.2) – which actually makes sense, since the window keeps growing until a timeout occurs, on which the window is reset, and set to the value *1* $\times$ *MSS*. In addition, the value of 'sstresh' is supposed to be set to half the window size before the packet loss started. However, this doesn't seem to be the case, as the initial value of the congestion window, is more like 200 bytes, and not 512 (which is the value of

MSS). The congestion window also changes size independently between the routers (for example, at *T = 0.4*, we can see that the congestion window for h3 and h4 is slightly different – even though both got the same initial MSS-value). I could not find any information wether or not the host/client can change the MSS-value themselves – however we know that *headers + MSS* $\leq$ *MTU*, which could suggest that if the MTU changes, the MSS-value could also change. Another thing worth noticing, is the high spike in the congestion window for h3 at *T = 0.6*.

Further on, by looking at Figure 2.3, we see that the SRRT (Smoothed round-trip time) changes according to the congestion window; every time the congestion window is reset (due to a timeout), the RTT is significantly faster (~10 ms). The RTT then grows in the same pace as the congestion window.

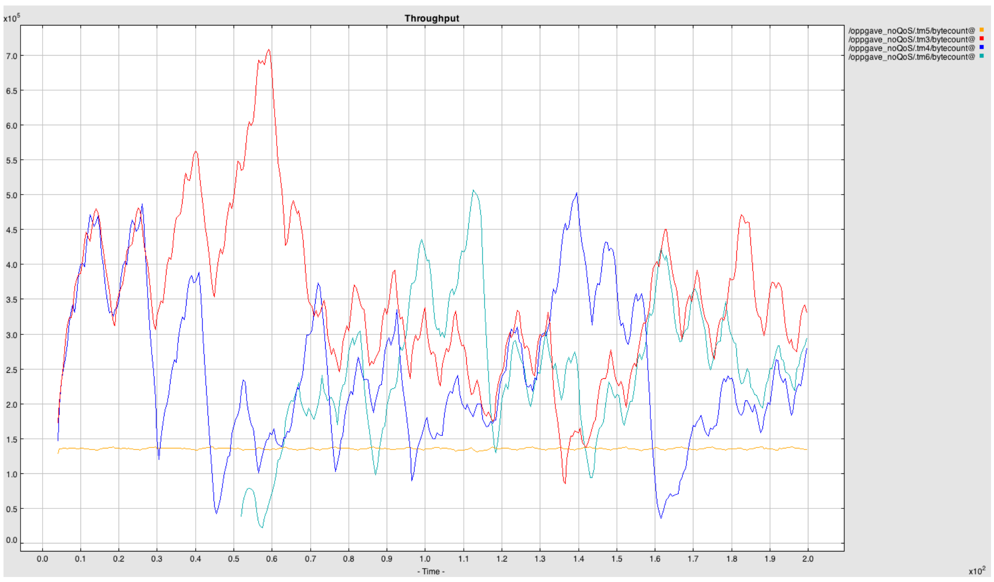

Figure 2.1: Throughput without QoS - Shows the throughput when no QoS is used.

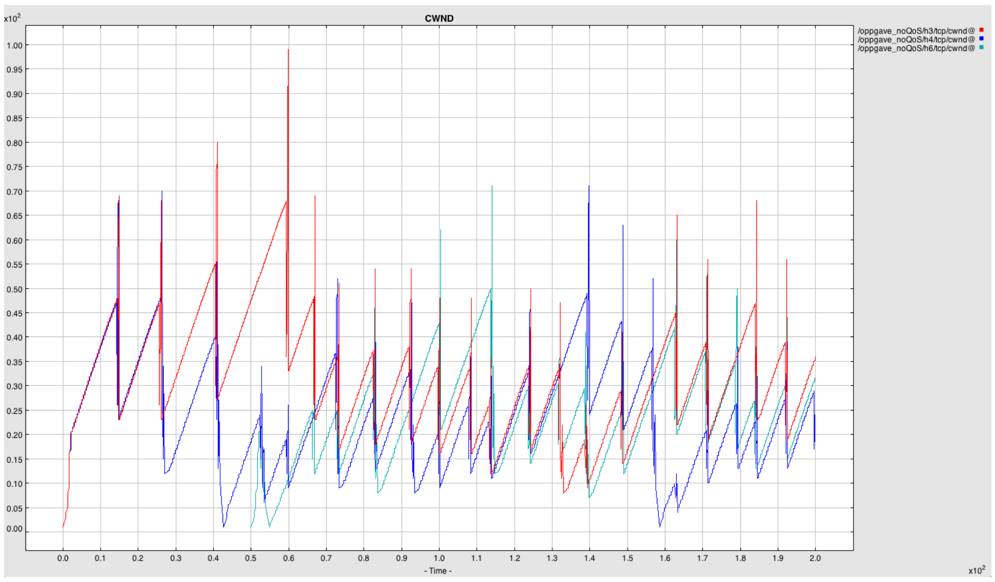

Figure 2.2: Congestion window without QoS - Shows how the congestion window changes when no QoS is used.

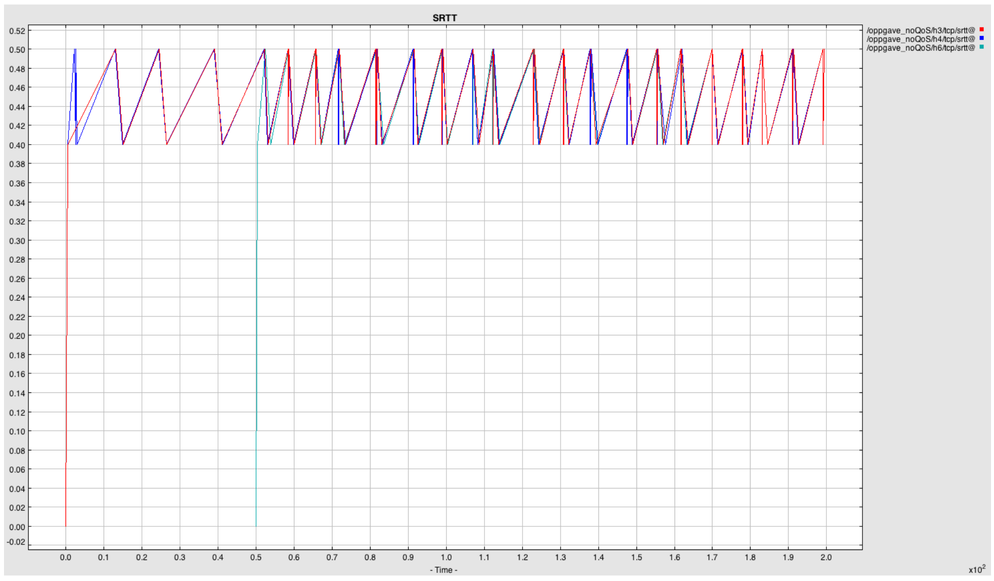

Figure 2.3: RTT without QoS - Shows how the (smoothed) RTT changes.

2.2 With QoS (Tail drop)

This simulation implements QoS, with tail drop for both EF/best efforts, and for AF (using WRR on AF1 and AF2). The links between the hosts/routers, are 100Mbps. The links to/from router n0, however, is set to 1Mbps. Router h3, h4 and h6 are all marked with AF11, while router h5 is marked with EF. Router n0 is set up with 4 queues; one best-effort queue (with a capacity of 8000 bytes), one EF-queue (with a capacity of 4000 bytes), one AF1-queue (with a capacity of 8000 bytes), and one AF2-queue (with a capacity of 8000 bytes). Before the AF1- and AF2-queues is sent to the Tx-queue, it's merged into one queue using WRR (Weighted Round Robin), where both queues is set a weight of 1.

The UDP-stream from h5 to h9, and the TCP-stream from h3 to h7, is started at the very beginning. After 70 seconds, the TCP-stream from h4 to h8 is started. After 130 seconds (from the start, not from when h4 to h8 was started), the TCP-stream from h6 to h10 is started.

After running the simulation, we can see from Figure 2.4 that the UDP-stream is not affected by the congestions that occur, which is not what happened when no QoS was involved. This is because the UDP-stream is marked with EF, and is therefore prioritized over the TCP-streams (which is marked with AF11).

Even though QoS is implemented, the streams still causes congestion, which causes the congestion window to be reset (as seen on Figure 2.5), which, in turn, also affects the throughput. This behaviour is just the same as we had in the simulation without QoS, however it's happening in a slightly more controlled manner – i.e. the reset of the congestion window "synchronizes" when there are more than one stream. This is because all the TCP-streams shares the same queue (since they are all marked with AF11), and when a congestion occurs, it occurs for all active streams at the same time.

It's worth noting the spike in the TCP-stream from h4 to h8 at the end of the run (at $T = 1.8$). I'm not really sure what's causing this, as there are no changes on h4, nor h8, at this point.

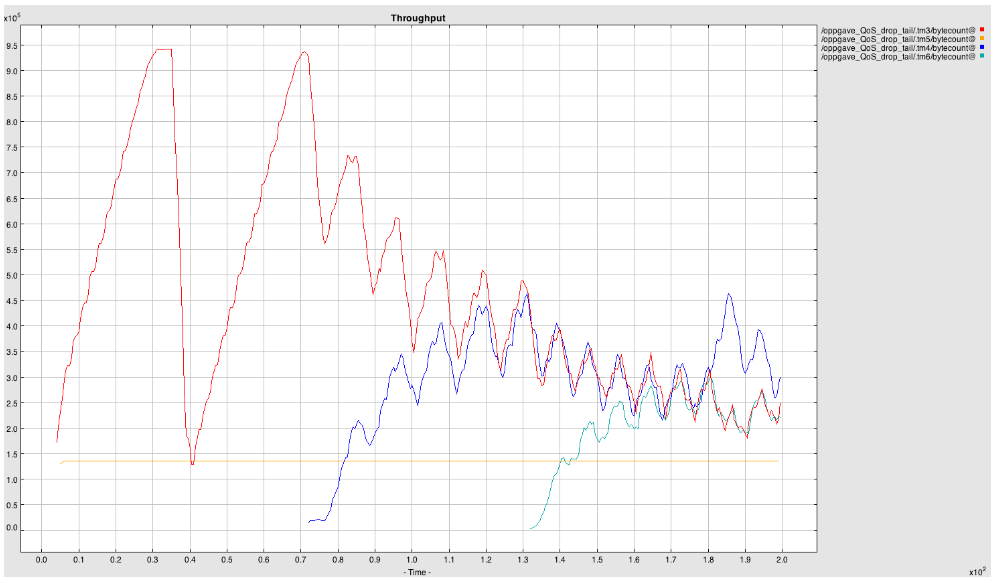

Figure 2.4: Throughput using QoS - Shows the throughput when QoS with tail drop is used.

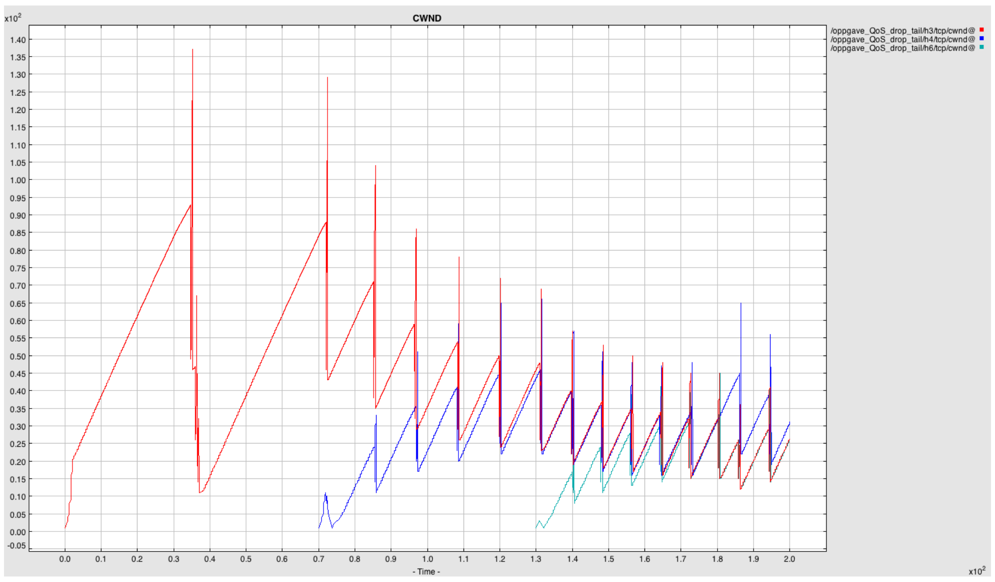

Figure 2.5: Congestion window using QoS - Shows how the congestion window changes when QoS with tail drop is used.

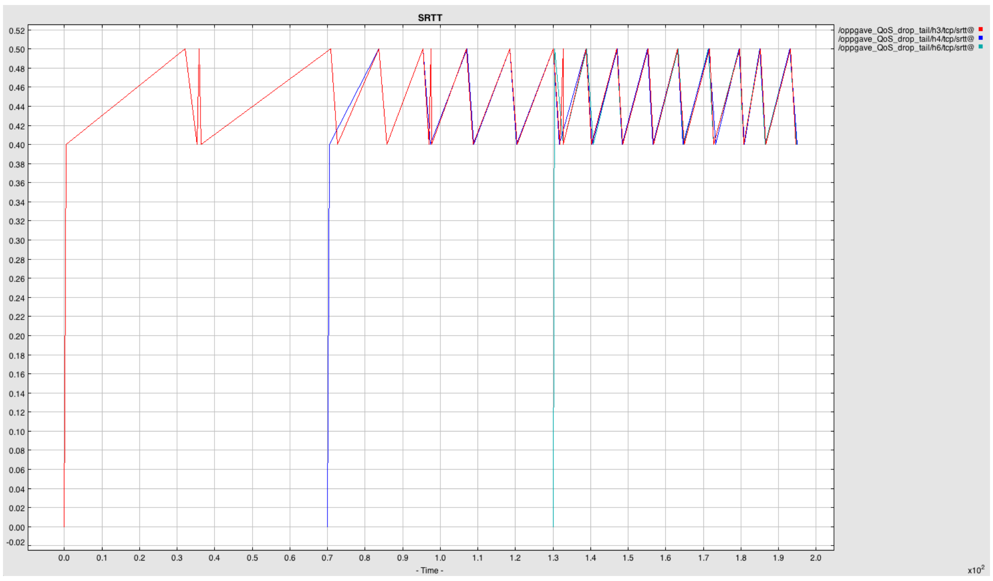

Figure 2.6: RTT with QoS - Shows how the (smoothed) RTT changes.

2.3 With QoS (RED)

This simulation implements QoS, with tail drop for EF/best efforts, and RED for AF. The links between the hosts/routers, are 100Mbps. The links to/from router n0, however, is set to 1Mbps. Router h3, h4 and h6 are all marked with AF11, while router h5 is marked with EF. Router n0 is set up with 3 queues; one best-effort queue (with a capacity of 8000 bytes), one EF-queue (with a capacity of 4000 bytes), and one AF-queue (with a capacity of 16000 bytes). The AF-queue is defined (in J-Sim) as a so-called "MQueue", which is an m-level queue generalized from RIO and 3-Color queue. It's a FIFO-queue with m queue logics, one for each specified level. The level-0 queue is the highest level queue, while the level-(m-1) is the lowest. It classifies incoming packets into m levels, and if a packets can be queued in a specific level, it's also enqueued to all the queues of lower levels without being rejected (and when it's dequeued, it's dequeued from it's classified level and all the lower levels).

The UDP-stream from h5 to h9, and the TCP-stream from h3 to h7, is started at the very beginning. After 70 seconds, the TCP-stream from h4 to h8 is started. After 130 seconds (from the start, not from when h4 to h8 was started), the TCP-stream from h6 to h10 is started.

After running the simulation, we can see from Figure 2.7 that the UDP-stream is not affected by the congestions that occur, which is the same as when using drop tail. Again, this is because the UDP-stream is marked with EF, and is therefore prioritized over the TCP-streams (which is marked with AF11).

As with drop tail, there's still congestion (so, QoS isn't *removing* congestion; it just makes the traffic flow more "smoothly"), which causes the congestion window to be reset (as seen on Figure 2.8). However, the congestion window resets more often than with drop tail. This is due to RED dropping packets before the queue is completely full.

Another thing is that the congestion window resets isn't as "synchronized" between the TCP-streams, as they where with drop tail. This is because we're not using WRR for the two AF-queues anymore, but rather two RED-levels/queues. This means that packets are classified differently, and since the two RED-queues has different parameters, packets flows accordingly.

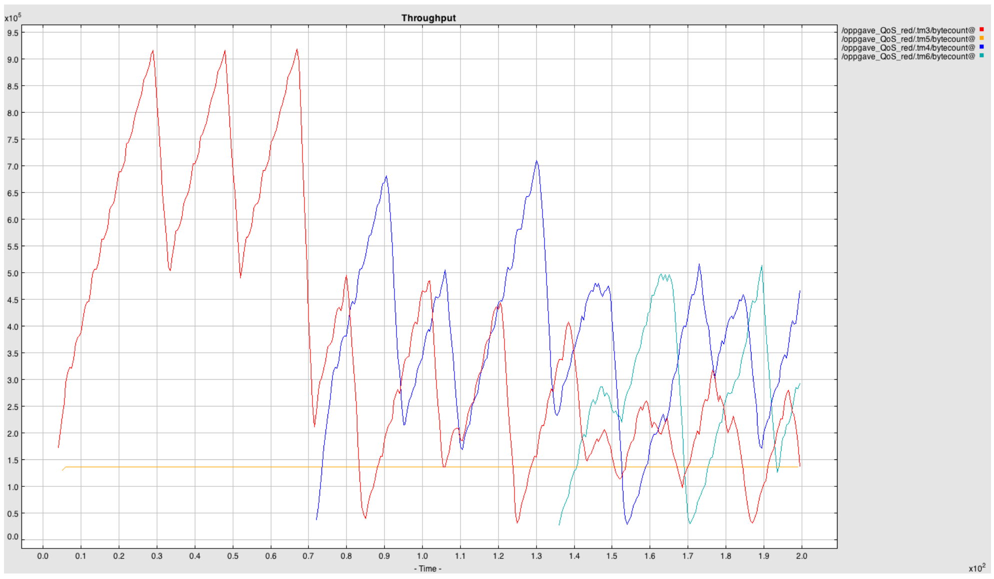

Figure 2.7: Throughput using QoS - Shows the throughput when QoS with RED is used.

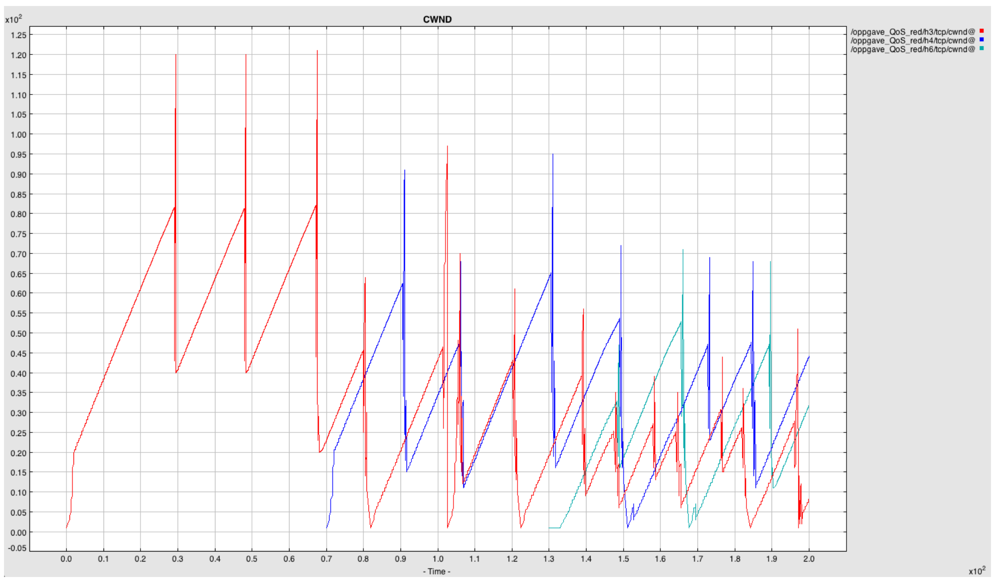

Figure 2.8: Congestion window using QoS - Shows how the congestion window changes when QoS with RED is used.

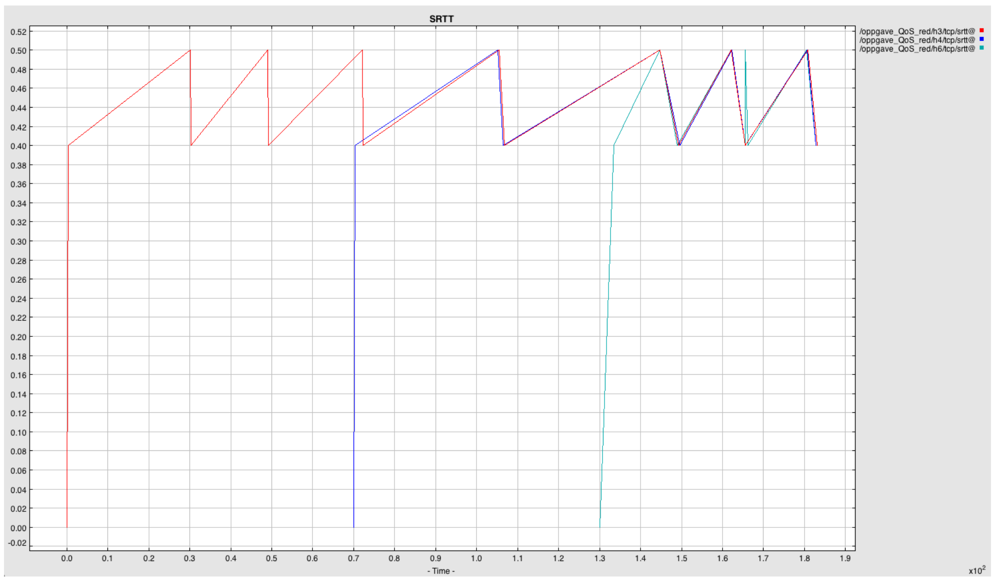

Figure 2.9: RTT with QoS - Shows how the (smoothed) RTT changes.

3

Conclusion

We can clearly see the effect of QoS, and what great job it does to mitigate congestions, while at the same time make sure important traffic is unaffected. QoS is really usefull in low-bandwidth settings, or f.ex. on asynchronous lines where voice applications is in use (or other applications with high requirements to packet loss or latency/jitter).

One thing I really didn't understand, is the high spikes in the congestion window just after it's been reset. It happens regardless if using QoS or not. I've been told that it could be due to fast retransmission, but I'm not really sure I understand *why* FRR is causing this behaviour.