Networks Lab 9 | Dabin Lee

1. What is the normal time required to download the webpage on h1 from h2?

   The normal time required to download the webpage on h1 from h2 is 1 second at the speed of 172KB/s.

2. What was your initial expectation for the congestion window size over time?

   I expected it to grow linearly ($CWS_i = 2 \times CWS_{i-1}$) and then enter an additive phase where congestion window size increases additively after reaching slow-start threshold. When a packet is dropped, CWS (Congestion Window Size) is reset to 1 MSS and slow-start threshold is set to half of the congestion window size before the packet loss.

3. After starting iperf on h1, did you observe something interesting in the ping RTT?

   The average ping RTT without running iperf was 30ms. After running iperf on h1, the average ping RTT was 700ms. After running iperf on h1, the ping RTT increased.

4. After starting iperf on h1, why does the web page take so much longer to download?

   With iperf running on h1, the traffic from iperf increases delay in the processing of the web page packets. Since the packets are processed in First-Come-First-Served basis, the packets before the wget traffic have to be processed first in order to process wget traffic and this is the main cause of the delay.

Networks Lab 9 | Dabin Lee

5. Please provide the figures for the first experiment (with qlen 100 and only one queue)
    a. Please comment on what you can see in the figures.

At about 10 seconds, iperf started and generated traffic and caused the quick increase in Figure 1. At about 86.5 seconds, wget started and completed at 91.5 seconds. When iperf's traffic in *Figure 3* hit 300KB/s, a packet loss occurs and the congestion window size is reduced as half. After a few seconds of additive increase in CWS, another packet was dropped and the CWS is reduced as half again. Additive increase continues until a packet is dropped at 100 packets / second. At 86.5 seconds, Figure 3 (iperf)'s CWS became 125 KB/s to share with Figure 2 (wget).

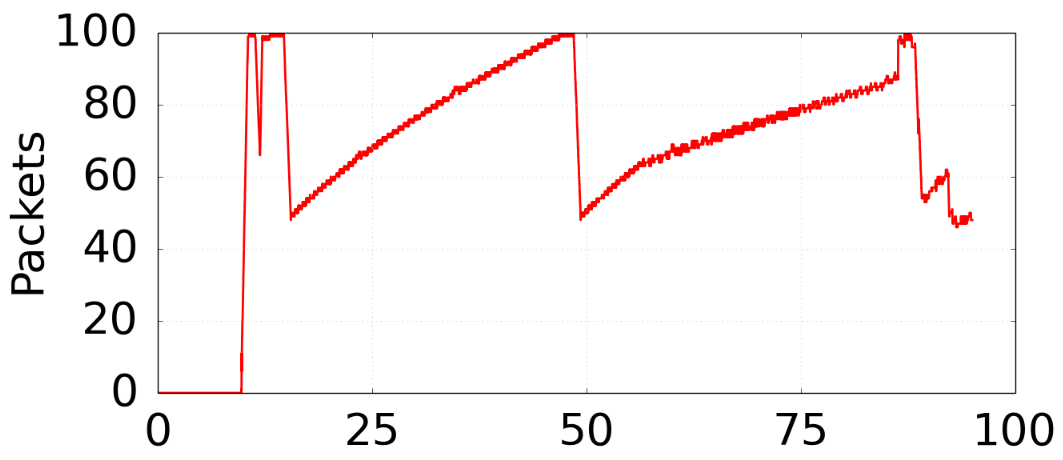
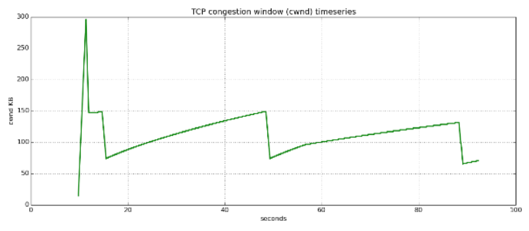


Figure 1 Experiment 1 Queue



2Figure 3 Experiment 1 CWND iperf

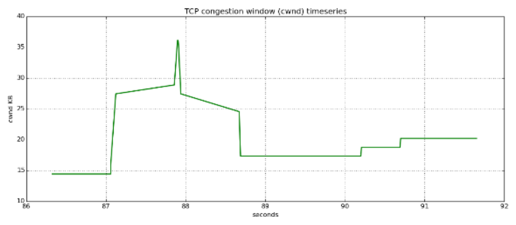


Figure 2 Experiment 1 CWND wget

Networks Lab 9 | Dabin Lee

6. Please provide the figures for the second experiment (with qlen 20 and only one queue)
    a. Please comment on what you can see in the figures, and what is different (and why)

The CWS for iperf seems to have way more fluctuation then it used to have in the previous experiment. This is due to the fact that the qlen was reduced to 20 instead of 100. With the shorter qlen, many of the packets are dropped and thus produces abrupt changes in the graph. Although it seems the "fair usage" problems seems to be improved, dropping many packets isn't an ideal solution to this problem.

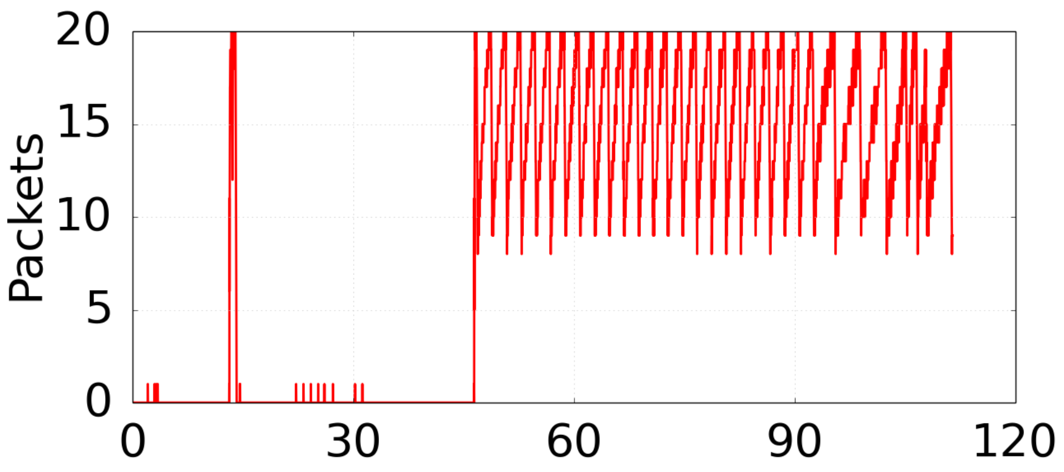


Figure 4 Experiment 2 Queue

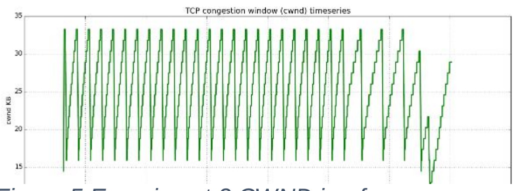


Figure 5 Experiment 2 CWND iperf

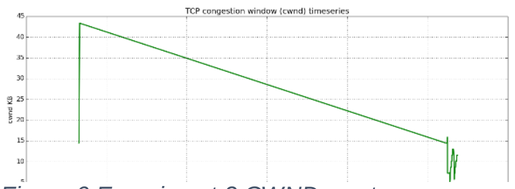


Figure 6 Experiment 2 CWND wget

Networks Lab 9 | Dabin Lee

7. Please provide the figures for the third experiment (with qlen 100 and two queues)
    a. Please comment on what you can see in the figures, and what is different

Compare to the first two experiments, this experiment's settings, qlen=100 and numberOfQue=2, seem to utilize the given capability better. With the scheduling that gives fair share of the link rate to each of the queues, short flow (wget) is not stuck somewhere because of the long flow (iperf). This scheme seems to more ideal in a sense that various services can be processed fairly, without one type of service taking the whole network capacity.

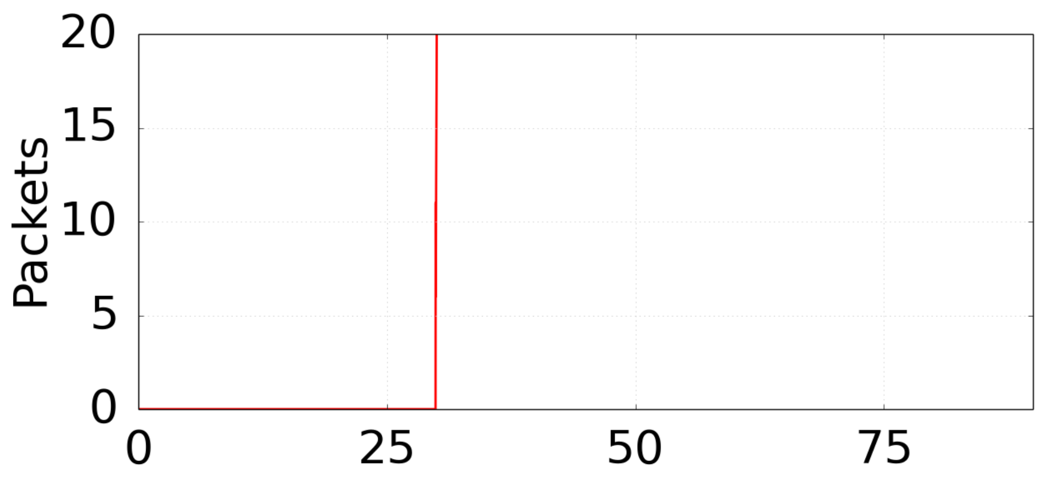


Figure 7 Experiment 3 Queue

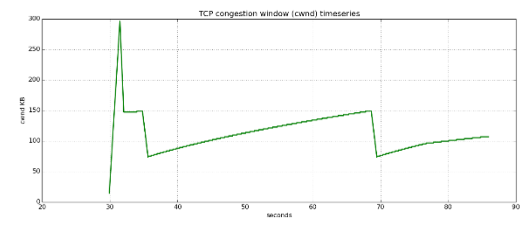


Figure 9 Experiment 3 CWND iperf

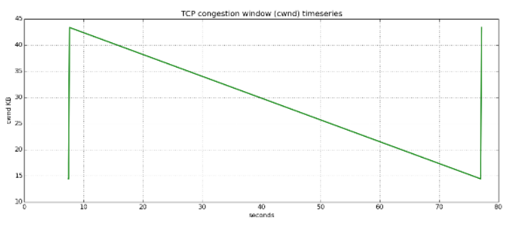


Figure 8 Experiment 3 CWND wget