CSE 232: Assignment 3

Name: Meetakshi Setiya
Roll no: 2019253

Question 1:

a.
The maximum expected throughput (theoretical) at the receiver node 2 is **5Mbps**. This is because the maximum throughput possible theoretically is equal to the bandwidth of the link. And here, the bandwidth is bottlenecked by the link between node1 and node2. So, even if the throughput from node n0 to n1 is 10Mbps, the receiver is going to get the packets at 5Mbps only because of the bandwidth between N1 and N2.

b.
-> For the entire network,

Total RTT delay for the packets to go from N0 to N2 = 20ms+30ms = 50×10^{-3} seconds.
Bandwidth of the network = 5Mbps = $5 \times (1024)^2 = 5242880$ bits.

Bandwidth delay product (BDP) = Bandwidth x Delay
$= 5242880 \times 50 \times 10^{-3}$
= 262144 bits

Actual length of a captured packet (as given in the pacp files generated by the ns3 simulation) = 1460 bytes
Therefore, BDP in terms of number of packets = 262144/8*1460= 22.44 ≈ 22 packets.

Note: if we take 5Mbps as 500Kb, the number of packets would be 21 approx.

-> If we find the bandwidth delay product for each link N0N1 and N1N2 separately,

BDP for N0N1 link : Bandwidth x RTT = 200Kb.
packet size = 1460 bytes
BDP for N0N1 = $200^* 10^3/8^*1460 = 17.12 \approx 17$ packets

BDP for N1N2 link : Bandwidth x RTT = 150Kb.
packet size = 1460 bytes
BDP for N0N1 = $150^* 10^3/8^*1460 = 12.28 \approx 12$ packets

c.
Average computed throughput of the TCP transfer (from n0 to n2) can be found directly in wireshark (tcp-example-2-0.pcapng) > Capture File Properties > Statistics

Average throughput = 4089×10^3 bits/sec or 4.089 Mbps

Another method:
Average computed throughput of the TCP transfer (from n0 to n2) can be calculated by dividing the bits exchanged with the duration as shown by wireshark.
From wireshark (tcp-example-2-0.pcapng) > statistics > conversations,

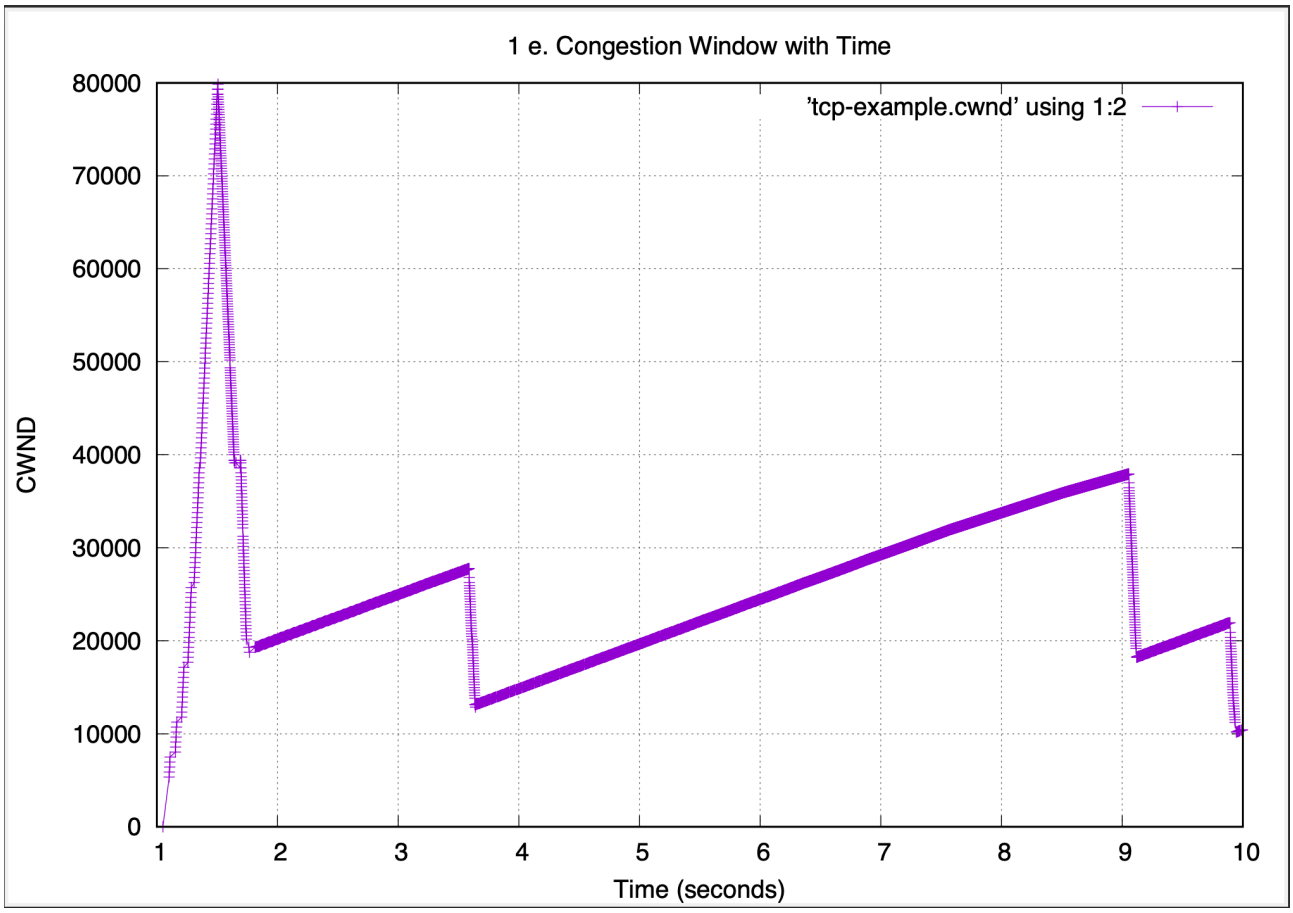
bytes exchanged in the TCP conversation = $4587 \times 10^3 = 8 \times 4587 \times 10^3$ bits.

duration = 8.9737 seconds

Average throughput = Bits exchanged/time taken = 8 x 4587 x 10^3 / 8.9744
= 4089283.127 bits/sec
= 4.089 Mbps

d.
No, the achieved throughput 4.089 Mbps is not equal to the expected throughput 5 Mbps. As a matter of fact, the achieved throughput is approximately 68% of the expected throughput (therefore, not comparable).
Throughput is a measure of actual, practical packet delivery rather than theoretical packet delivery. Maximum expected throughput (bandwidth) tells how much data can be potentially sent and received in a second whereas throughput is the amount of data that is actually being exchanged in a second. Factors like latency, transmission errors (error rate), jitter etc often limits the throughput to less than its maximum theoretical value.
Since we introduced an error model into our program and there are latencies introduced in our network which maybe be some of the reasons why the actual throughput is less than the maximum expected throughput.

e.
I used Gnuplot to plot the congestion window with time obtained by running a Gnuplot script. The file used was the tcp-example.cwnd file generated by the tcp-example.cc simulation.



The script used the plot the graph was:

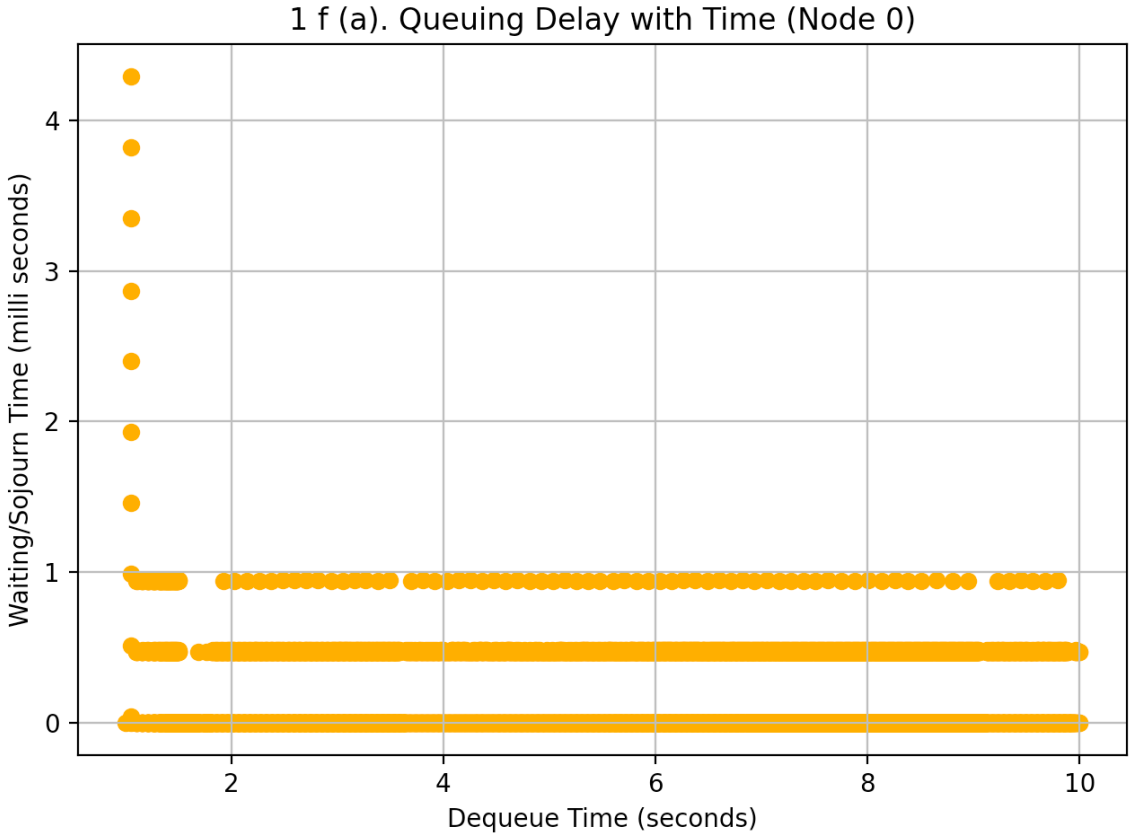
```
set term postscript eps color
set output 'cwnd.eps'
```

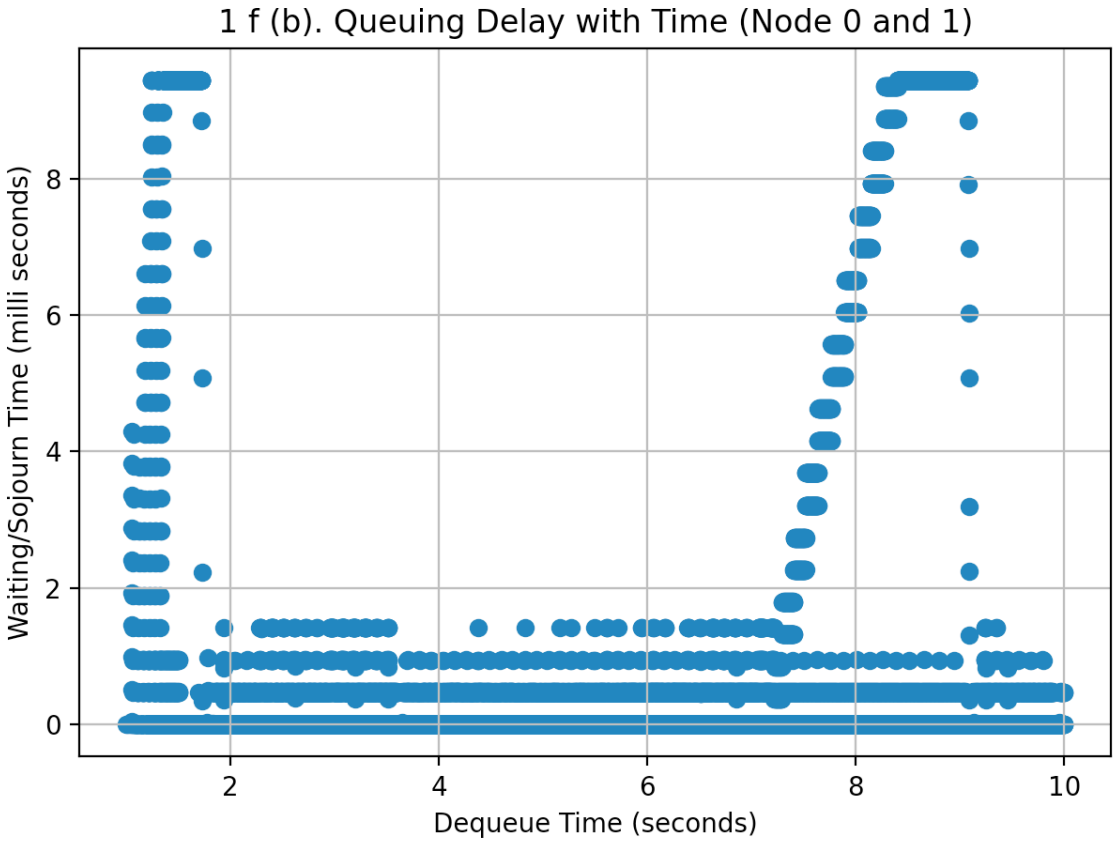
```
set title "1 e. Congestion Window with Time"
set ylabel 'CWND'
set xlabel 'Time (seconds)'
set grid
plot 'tcp-example.cwnd' using 1:2 with linespoints
```

f.

I used a python script to make the Queuing delay graph using tcp sequence number and node number as indices from the tcp-example.tr file. As was mentioned in the assignment instructions, the parts of interest for us are mainly the entries corresponding to a packet getting enqueued and dequeued at the queue of node N0.

Figures 1f (a) and (b) show graphs showing queuing delay only at Node 0 and at Node0 and Node 1 together respectively.





g.

Note that since we are only talking about congestion window at Node 0, I will be comparing the graphs of congestion window with time (1e) with queuing delay graph at node 0 (1f (a)) only.

Yes, the plots 1e and 1f(a) are related. This is the TCPReno variant. If we look at the plot 1e, we see that for the 1 to about 1.5 second mark, the TCP socket is in the slow start phase. In this phase, the window size increases exponentially after each successful ACK. The congestion window thus shoots up to about 80000. If we look at the queuing delay graph for this interval, we see that the queuing delay also increases greatly during this period. This is because more packets are queued before they are transmitted and have to spend incrementally more time in the queue. After the threshold 'ssthresh' is reached or packet loss starts happening, near the 1.5 second mark, the packets begin to get dropped and near the 2 second mark, TCP starts the congestion avoidance phase. Now, the window size increases by one MSS after each successful ACK when TCP again starts grabbing back the bandwidth. The congestion window then seems to be rising and falling after receiving 3 duplicate ACKs in a row or reaching the threshold 'ssthresh' meaning that TCP has entered the fast recovery phase at times, reducing the window size to half. Nevertheless, the network/receiver can keep up with this pace now and the queue waiting time remains pretty much constant (below the 1 millisecond mark) up till the end of the simulation.

In other words, we see the window size increasing progressively after each successful ACk but the queueing delay is pretty much unaffected because the window size is increasing linearly instead of exponentially like during the slow start phase.

PS: DropTail drops arriving packets when queue is full regardless of flow or importance.

Question 2:

a.

Average computed throughput of the TCP transfer (from n0 to n2) can be found directly in wireshark (tcp-example2-2-0.pcapng) > Capture File Properties > Statistics

Average throughput = 4230 x 10^3 bits/sec or 4.230 Mbps

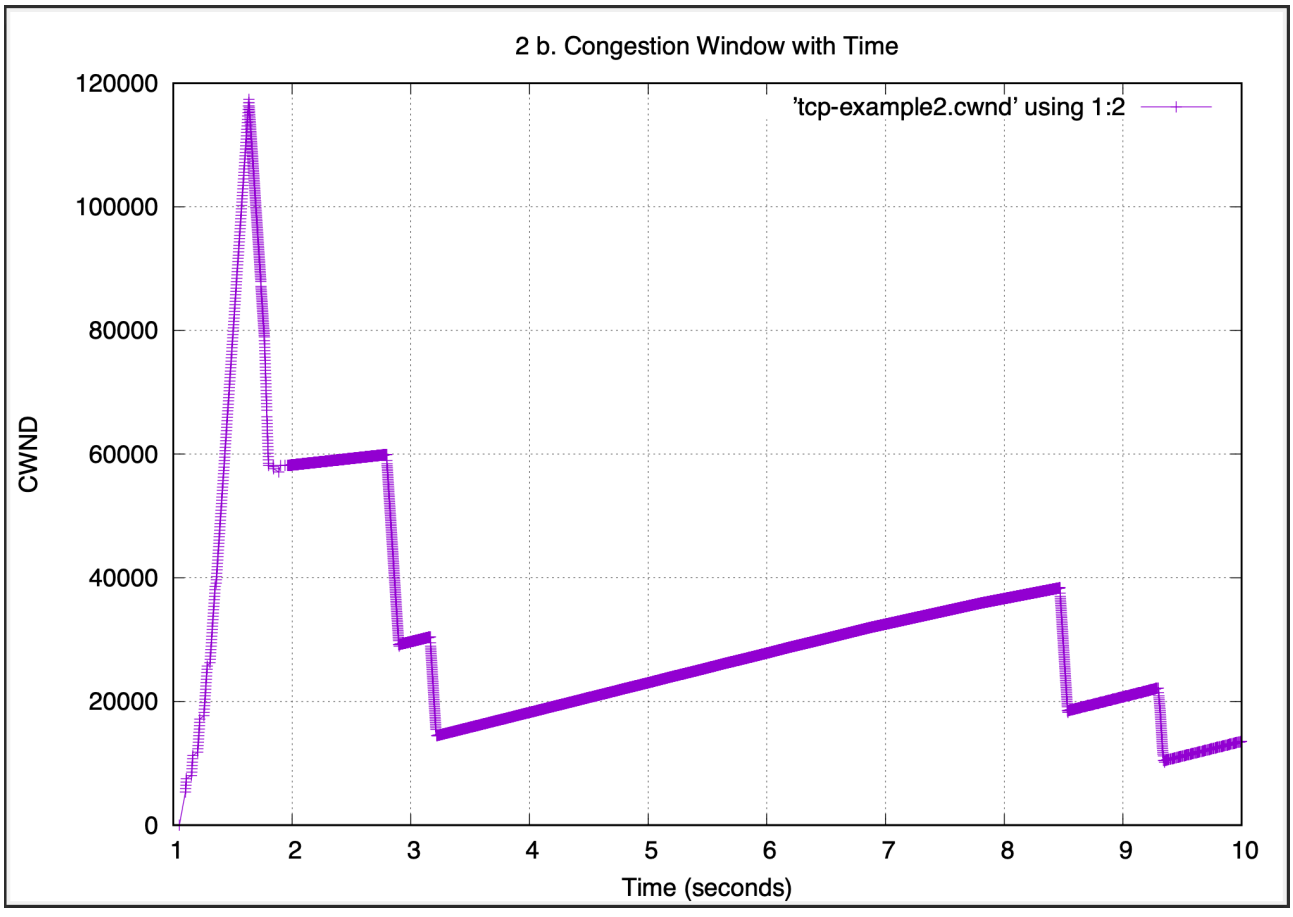
Another method:
Average computed throughput of the TCP transfer (from n0 to n2) can be calculated by dividing the bits exchanged with the duration as shown by wireshark.
From wireshark (tcp-example2-2-0.pcapng) > statistics > conversations,

bytes exchanged in the TCP conversation = 4745 x 10^3 = 8 x 4745 x 10^3 bits.
duration = 8.9746 seconds

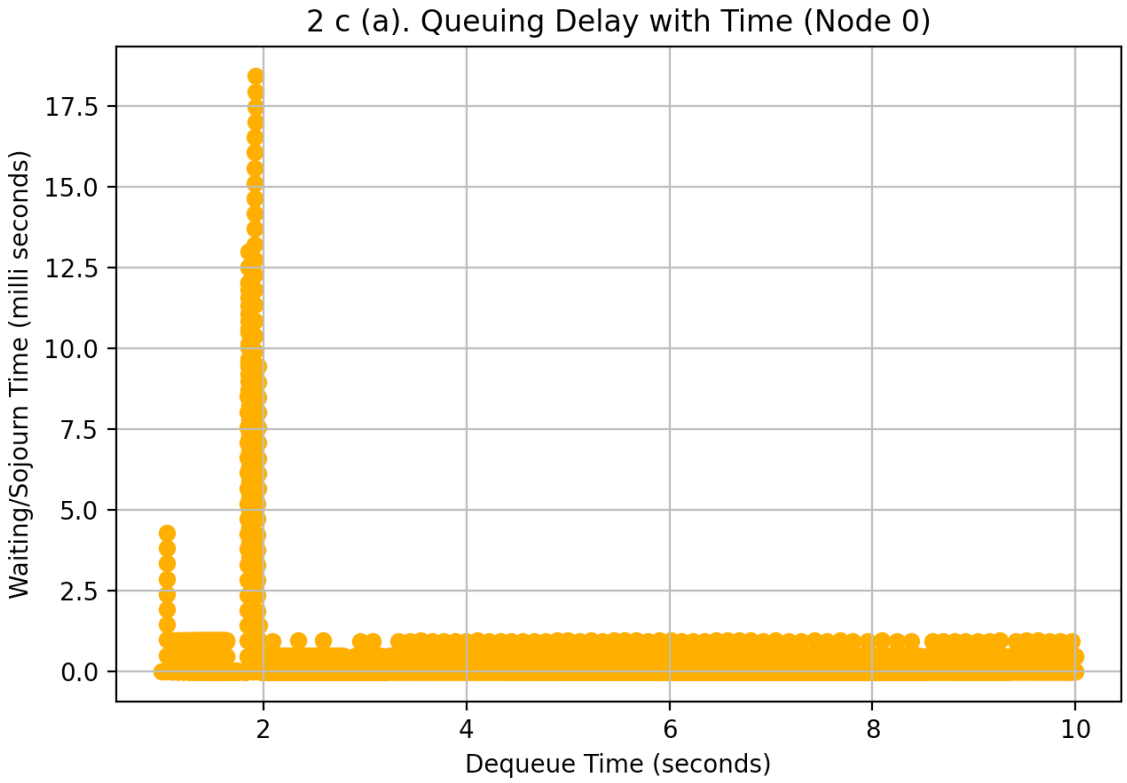
Average throughput = Bits exchanged/time taken = 8 x 4745 x 10^3 / 8.9746
= 4229714.97 bits/sec
= 4.230 Mbps

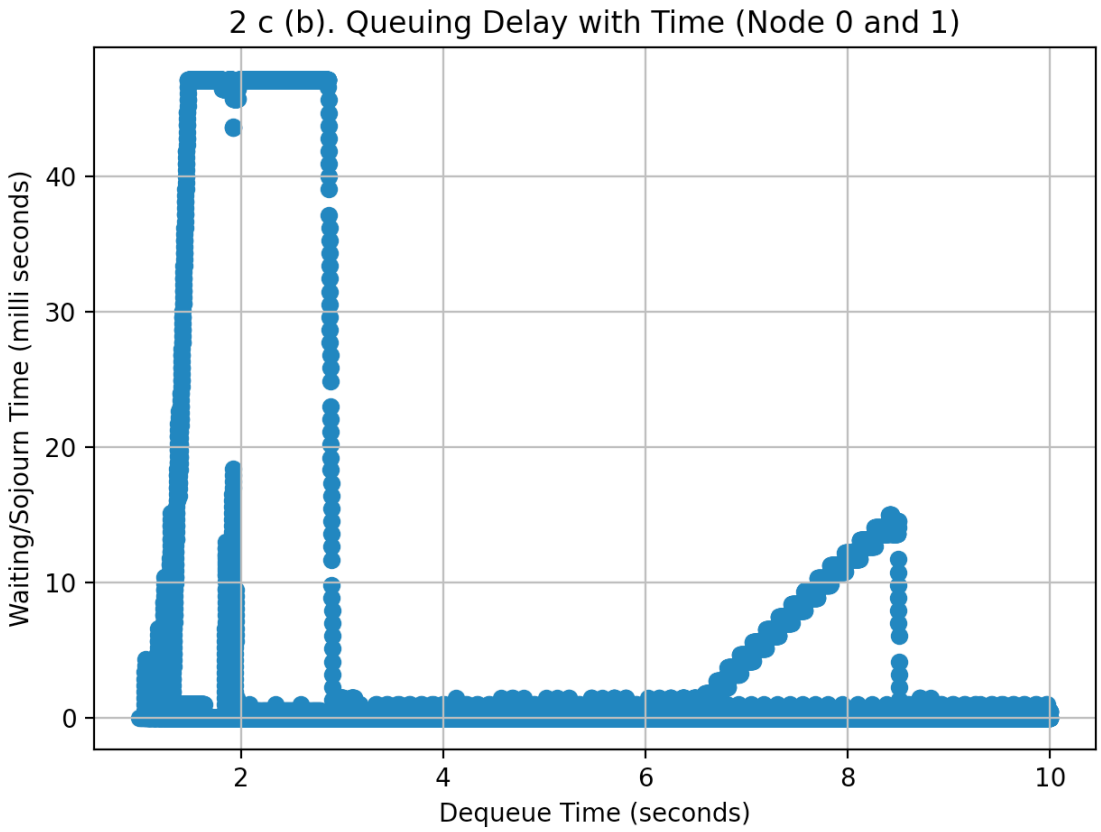
b.
I used Gnuplot to plot the congestion window with time obtained by running a Gnuplot script. The file used was the tcp-example2.cwnd file generated by the tcp-example.cc simulation.
The script used the plot the graph was:

```
set term postscript eps color
set output 'cwnd2.eps'
set title "2 b. Congestion Window with Time"
set ylabel 'CWND'
set xlabel 'Time (seconds)'
set grid
plot 'tcp-example2.cwnd' using 1:2 with linespoints
```



c.
I used a python script to make the Queuing delay graph using tcp sequence number and node number as indices from the tcp-example2.tr file. As was mentioned in the assignment instructions, the parts of interest for us are mainly the entries corresponding to a packet getting enqueued and dequeued at the queue of node N0.
Figures 2c (a) and (b) show graphs showing queuing delay only at Node 0 and at Node0 and Node 1 together respectively.





d.

Comparing the plots 1e and 2b brings to light the effect of increasing queue size on the congestion window size progressively with time. From a single glance, it is visible that the congestion window in 1e reaches almost 120000 which is greater than the 80000 mark reached in 1e. This happens because in the slow start phase, when the window size increases exponentially after each successful ACK, a larger queue is capable of holding more packets that are waiting to get transferred, therefore enabling TCP to transmit more data from Node0. Since CWND is a fair estimate of the number of packets in flight, a larger queue, with a greater number of packets at the ready is liable to have a larger congestion window. Now, once the sender (Node0) starts receiving repeated ACKs or when ssthresh is reached, the fast recovery phase kicks in. One insight is that the queue size actually does play a role in determining this network “ceiling” after which packets start getting lost.

The congestion avoidance phase kicks in a little later in 2b. After the slow start phase in both, the congestion window is reduces in size for both 1e and 2b to half in the fast recovery phase. Packets stop dropping near the 20000 congestion window mark in 1e and 60000 congestion window mark in 2b. Now, TCP starts the congestion avoidance phase and the window size is increased linearly by 1 MSS after every successful ACK. In plot 2b, between the 2 and 3 second mark, the congestion window increases in size as TCP starts grabbing the bandwidth back, but then packet dropping starts, probably because of the presence of unacknowledged packets or errors on the link.

The reason that we see more drops in 2b than in 1e is owing to the greater queue size in 2b which lines up a lot more packets waiting to be sent down the link and the outflow might be bottlenecking.

It has to be noted that the throughput has increased on increasing the queue size but queuing delay has increased as well.

Question 3:

a.
Average computed throughput of the TCP transfer (from n0 to n2) can be found directly in wireshark (tcp-example3-2-0.pcapng) > Capture File Properties > Statistics

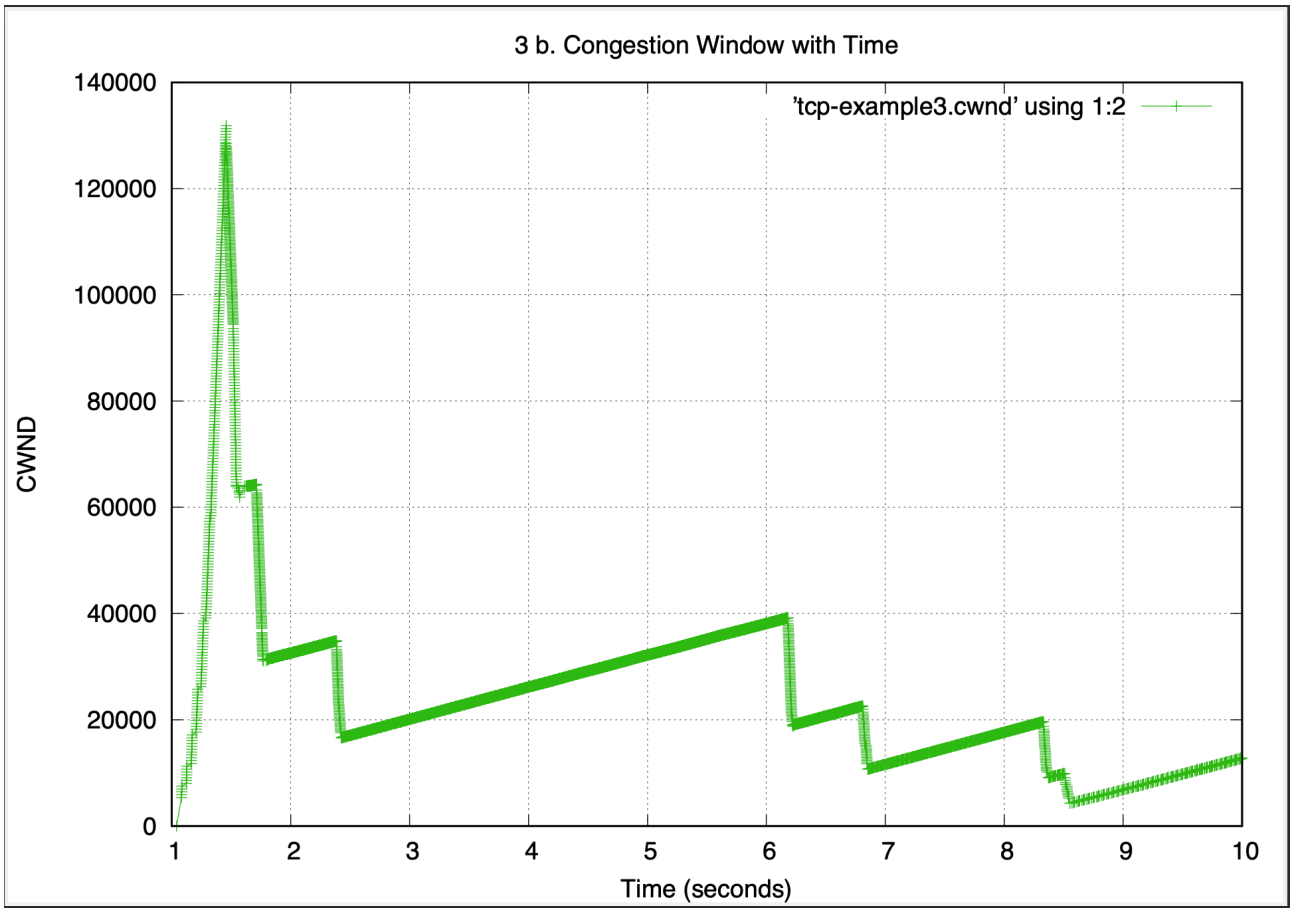
Average throughput = 4958 x 10^3 bits/sec or 4.958 Mbps

Another method:
Average computed throughput of the TCP transfer (from n0 to n2) can be calculated by dividing the bits exchanged with the duration as shown by wireshark.
From wireshark (tcp-example3-2-0.pcapng) > statistics > conversations,

bytes exchanged in the TCP conversation = 5562 x 10^3 = 8 x 5562 x 10^3 bits.
duration = 8.9746 seconds

Average throughput = Bits exchanged/time taken = 8 x 5562 x 10^3 / 8.9746
= 4957992.557 bits/sec
= 4.958 Mbps

b.
I used Gnuplot to plot the congestion window with time obtained by running a Gnuplot script. The file used was the tcp-example3.cwnd file generated by the tcp-example.cc simulation.



The script used the plot the graph was:

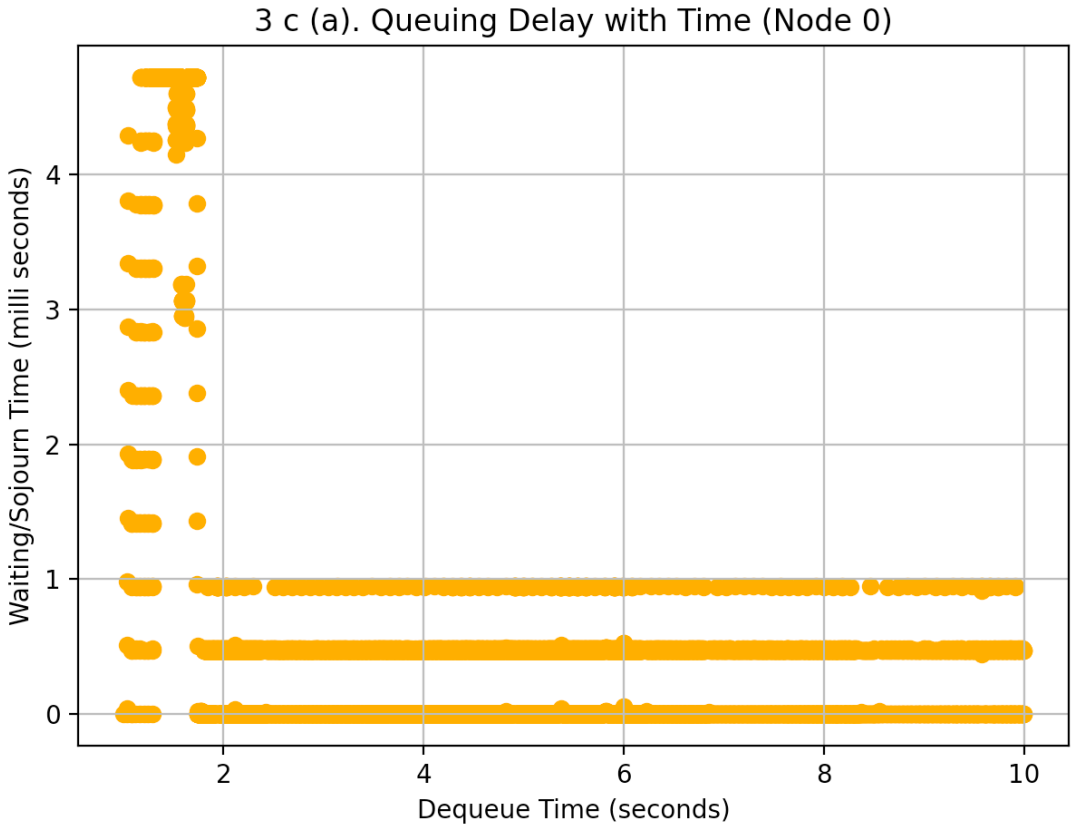
```
set term postscript eps color
```

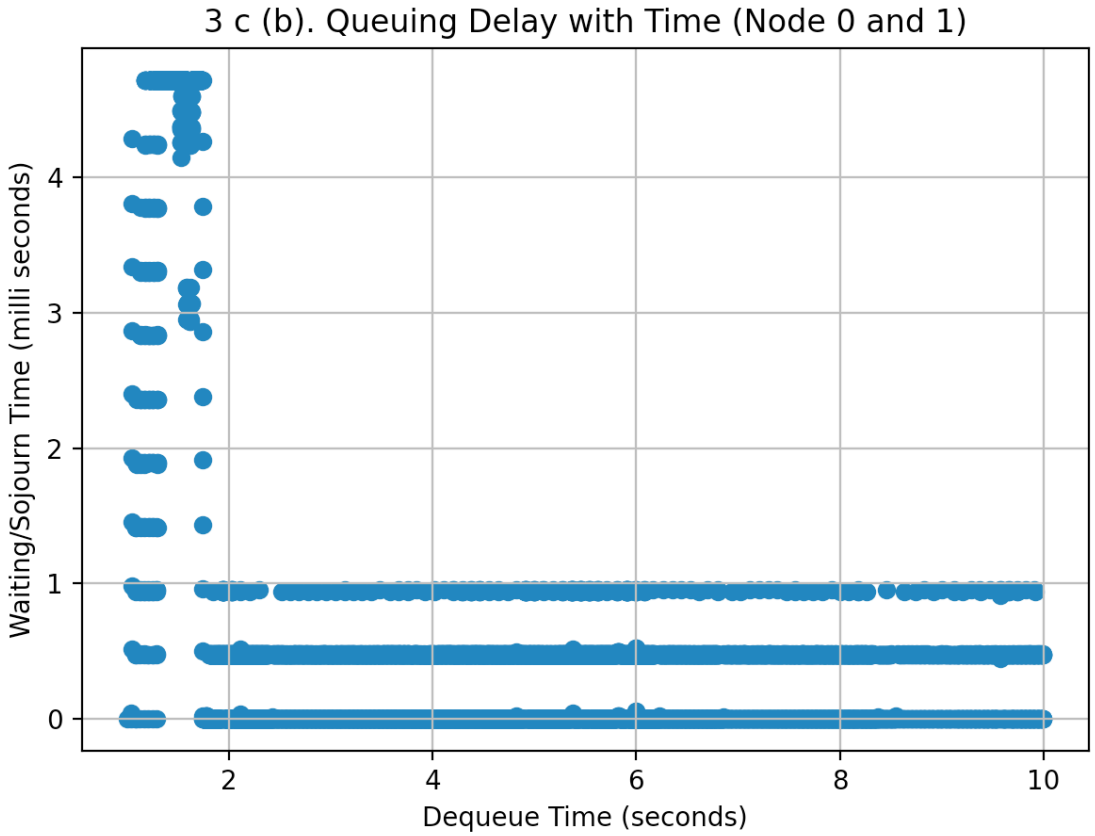
```
set output '3b.eps'
set title "3 b. Congestion Window with Time"
set ylabel 'CWND'
set xlabel 'Time (seconds)'
set grid
plot 'tcp-example3.cwnd' using 1:2 with linespoints lt rgb “#30ba14"
```

c.

I used a python script to make the Queuing delay graph using tcp sequence number and node number as indices from the tcp-example3.tr file. As was mentioned in the assignment instructions, the parts of interest for us are mainly the entries corresponding to a packet getting enqueued and dequeued at the queue of node N0.

Figures 3c (a) and (b) show graphs showing queuing delay only at Node 0 and at Node0 and Node 1 together respectively.





d.

Note that since we are only talking about congestion window at Node 0, I will be comparing the graphs of queuing delay vs time at node 0 (1f (a)) (3c (a)) only.

The bandwidth-delay product for the link N1N2 in Part 1 is 0.075 Mbps. In Part 3 though, the bandwidth-delay product for the link N1N2 is 0.1 Mbps. This effects the overall network too since the bandwidth bottleneck has increased. Ultimately, this means that more data can be transmitted by a sender through the link at the same rate as it is arriving because of the same RTT and bandwidths of the two links. In the slow start phase, the queuing delay peak achieved in 3c(a) is less than that achieved in 1f owing to faster acknowledgement and transmission of packets in the network because a comparatively higher number of bits can fill up the link N1N2 now. After the packets start dropping marking the end of the slow start phase, the waiting time is pretty much stabilised for both configurations in the TCP congestion avoidance phase. In other words, the bandwidth does not really matter anymore as the queue is kind of stable now and the network is not brimming with packets in queue at N0 waiting to be sent but hampered by the limitations of the link.

Additionally, the average throughput for the increased bandwidth-delay product configuration is also higher. Simply because the network speed in the slow start phase would be higher for 3c(a) and not bottlenecked by the 5Mbps link and higher delay as was in 1f(a). This makes a considerable difference in the total number of packets exchanged and hence the throughput.

Question 4:

a.

Average computed throughput of the TCP transfer (from n0 to n2) can be found directly in wireshark (tcp-example-2-0.pcapng) > Capture File Properties > Statistics

Average throughput = 4405 x 10^3 bits/sec or 4.405 Mbps

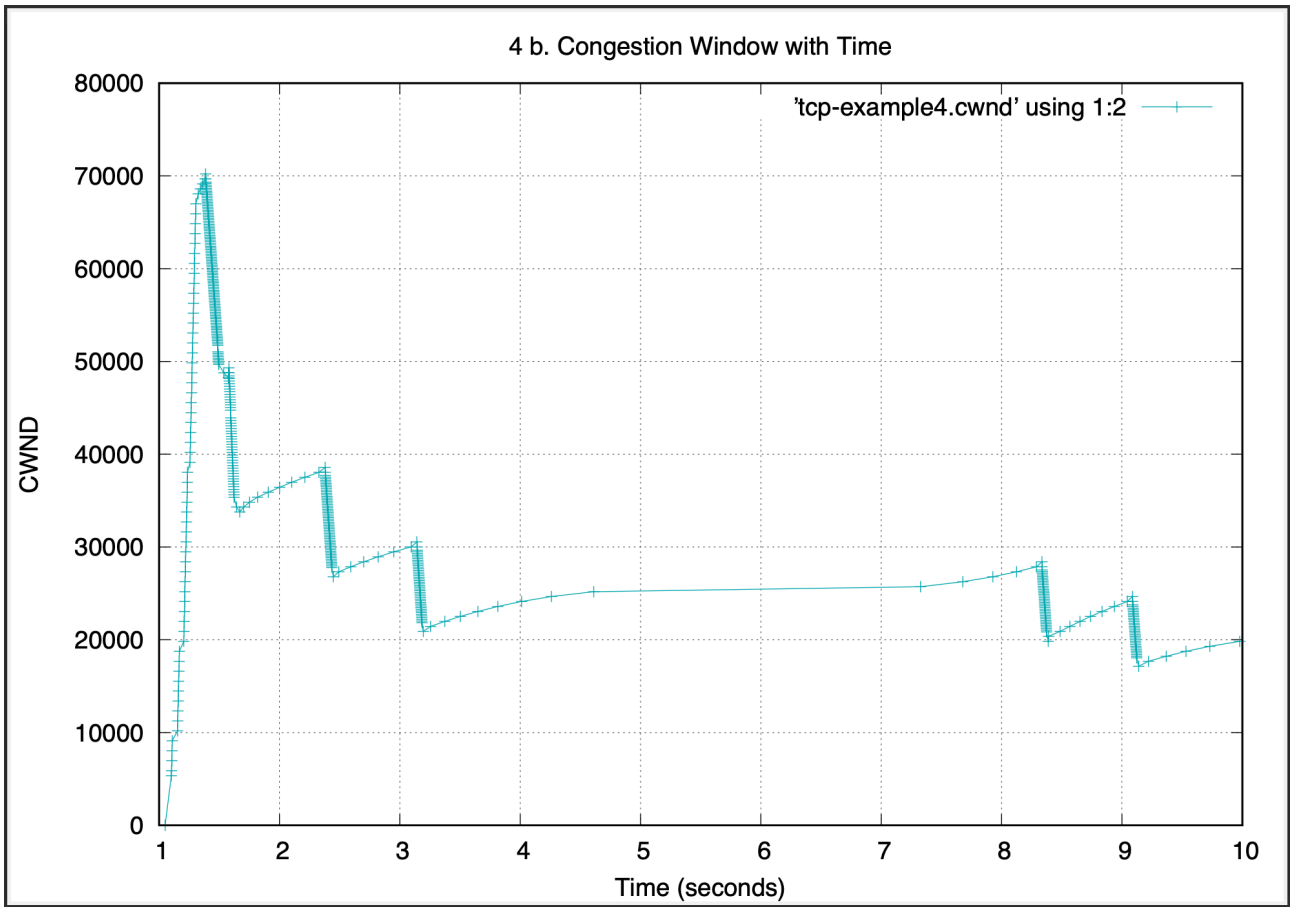
Another method:
Average computed throughput of the TCP transfer (from n0 to n2) can be calculated by dividing the bits exchanged with the duration as shown by wireshark.
From wireshark (tcp-example4-2-0.pcapng) > statistics > conversations,

bytes exchanged in the TCP conversation = 4941 x 10^3 = 8 x 4941 x 10^3 bits.
duration = 8.9730 seconds

Average throughput = Bits exchanged/time taken = 8 x 4941 x 10^3 / 8.9730
= 4405215.647 bits/sec
= 4.405 Mbps

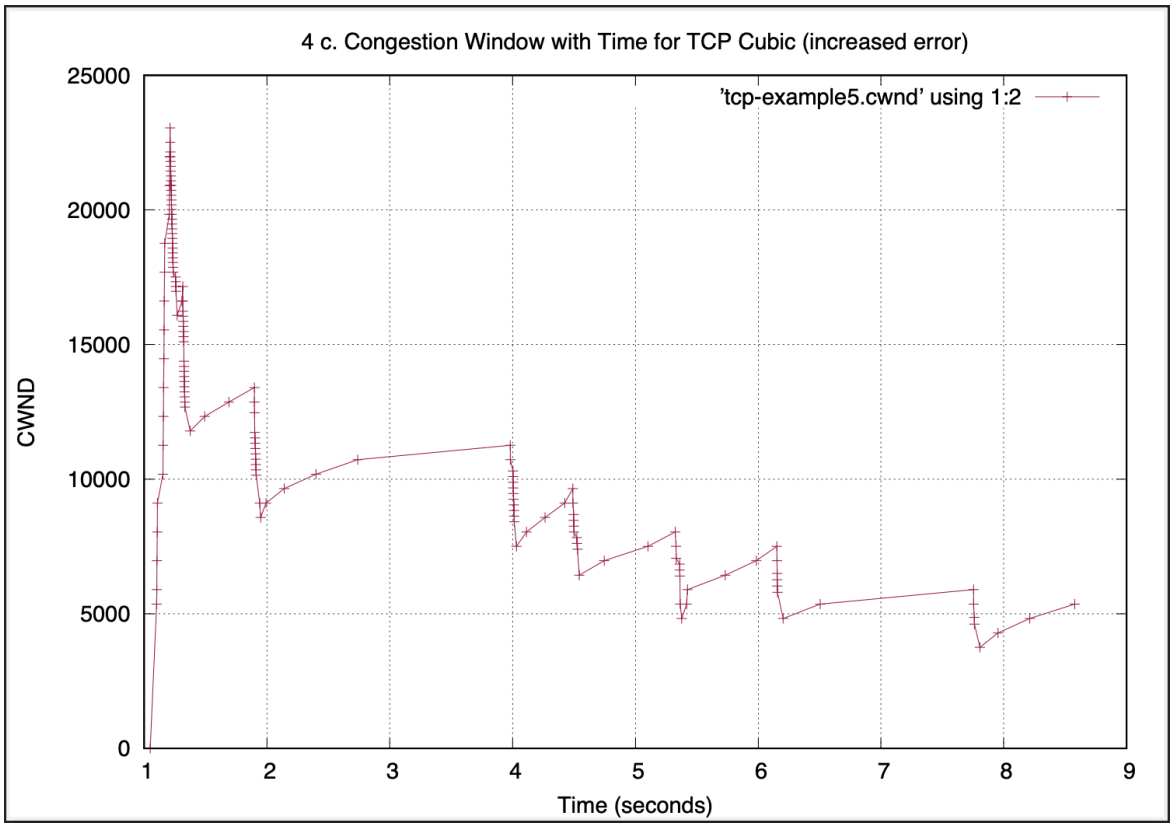
b.
I used Gnuplot to plot the congestion window with time obtained by running a Gnuplot script. The file used was the tcp-example3.cwnd file generated by the tcp-example.cc simulation.
The script used the plot the graph was:

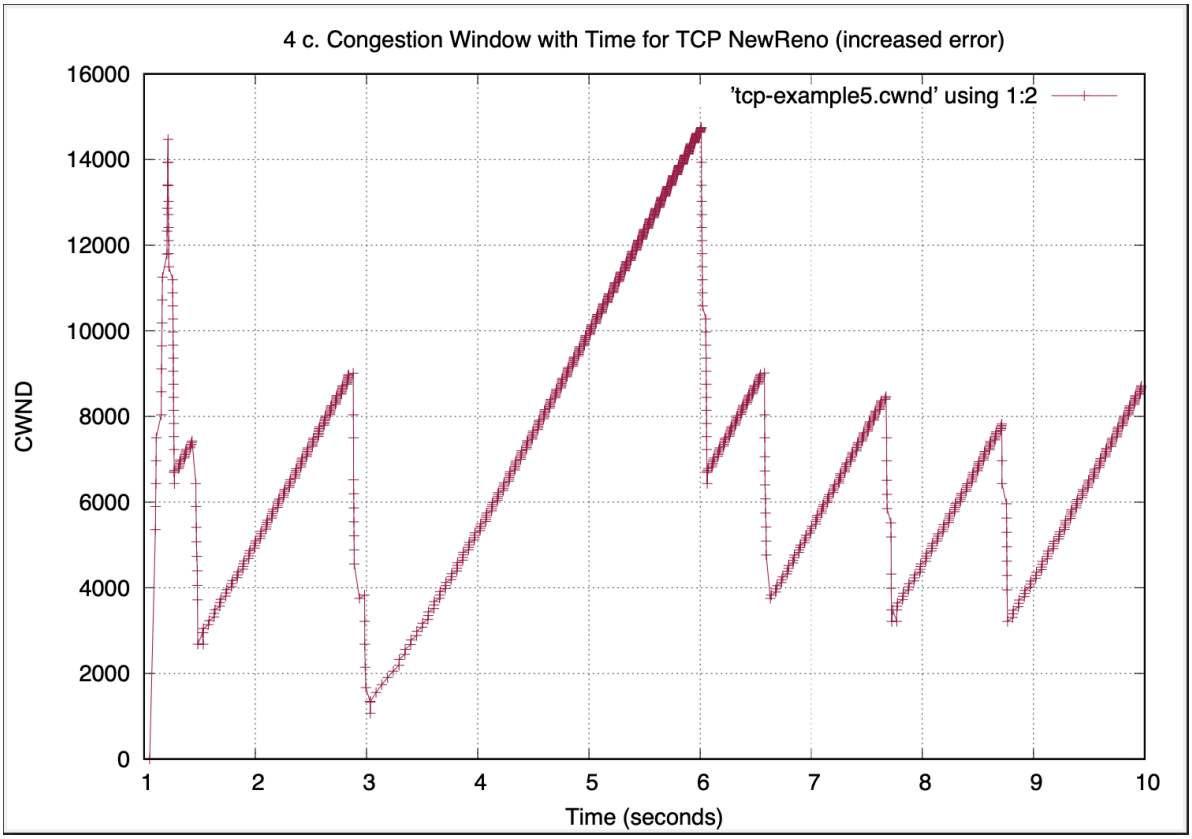
```
set term postscript eps color
set output '4b.eps'
set title "4 b. Congestion Window with Time"
set ylabel 'CWND'
set xlabel 'Time (seconds)'
set grid
plot 'tcp-example4.cwnd' using 1:2 with linespoints lt rgb “#0badb5"
```



c.

The difference between the CWND vs Time graphs of the TCP New Reno and TCP Cubic variants will be more visible on increasing the error rate to 0.00001. The respective graphs for TCP NewReno and Cubic are given below:





Now, the differences between the two TCP variants during the slow start, congestion avoidance, and fast recovery phases can easily be pointed out:

Slow start: The slow start phase for both TCP NewReno and Cubic are the same, increasing congestion window only at the reception of ACK. The window size here increases exponentially for both variants until the threshold ssthresh is reached or 3 repeated ACKs are received. The difference in peaks in the two graphs through could be because of the network errors or some difference in the underlying exponential algorithm responsible for increasing the window size.

Congestion Avoidance: The congestion avoidance phase for both variants differs. TCP NewReno employs a linear growth function- increasing the window size by 1 MSS after each successful acknowledgement (ACK). TCP Cubic however, changes the window update algorithm and uses a cubic function to do the job instead. Upon receiving an ACK during this phase, TCP Cubic computes the window growth rate during the next RTT period using `W_cubic(t) = C*(t-K)^3 + W_max`

K is the time period that the above function takes to increase the current window size to W_max if there is no further loss event:

`K = cubic_root(W_max*(1-beta_cubic)/C)`
where beta_cubic is the CUBIC multiplication decrease factor.

Additionally, depending on the current window size, TCP cubic runs in TCP Friendly, Concave and Convex region thus explaining the inter-variations in the curve for TCP Cubic during different congestion avoidance phases. The congestion avoidance phases in TCP NewReno's graph are linear, with the same slope and without any other variation.

Fast Recovery: In the fast recovery phase, TCP Cubic and TCP NewReno differ in their window decrease multiplicative factors. TCP Cubic changes the window size to 0.7 times of

its original size after each duplicate ACK whereas TCP NewReno sets this factor at 0.5 and reduces the window size after each duplicate ACK received.

Additional observations: The average throughput obtained by using TCP cubic was higher than one obtained after using TCP NewReno. This is because TCP Cubic uses a moore aggressive window growth function than NewReno variant leading to a lesser packet loss rate. This indicates that TCP Cubic can effectively utilise the spare bandwidth left unused by NewReno without taking away much bandwidth from the already existing flows.

(Reference: https://tools.ietf.org/id/draft-ietf-tcpm-cubic-05.html#rfc.section.4.1)