Advanced Networking 2018

Lab #1: TCP Congestion Control

Report

GROUP: 12

Authors:
Adrien Raulot, adrien.raulot@os3.nl
Kotaiba Alachkar, kotaiba.alachkar@os3.nl
University of Amsterdam

Q1.1 Plot a graph showing CWND versus time from 0.0s to 100.0s.

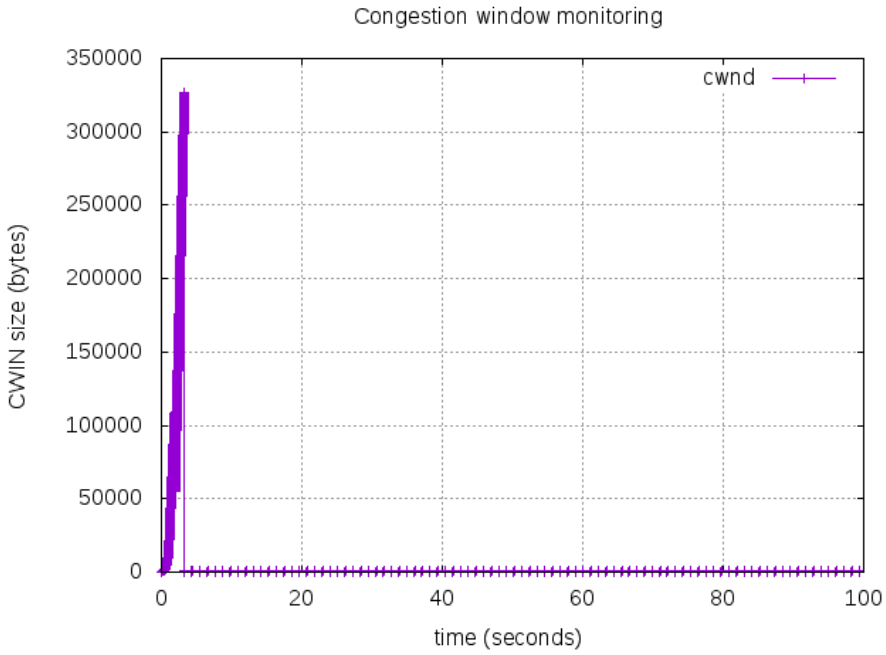

Figure 1: Reno CWND time 0.0s to 100.0s

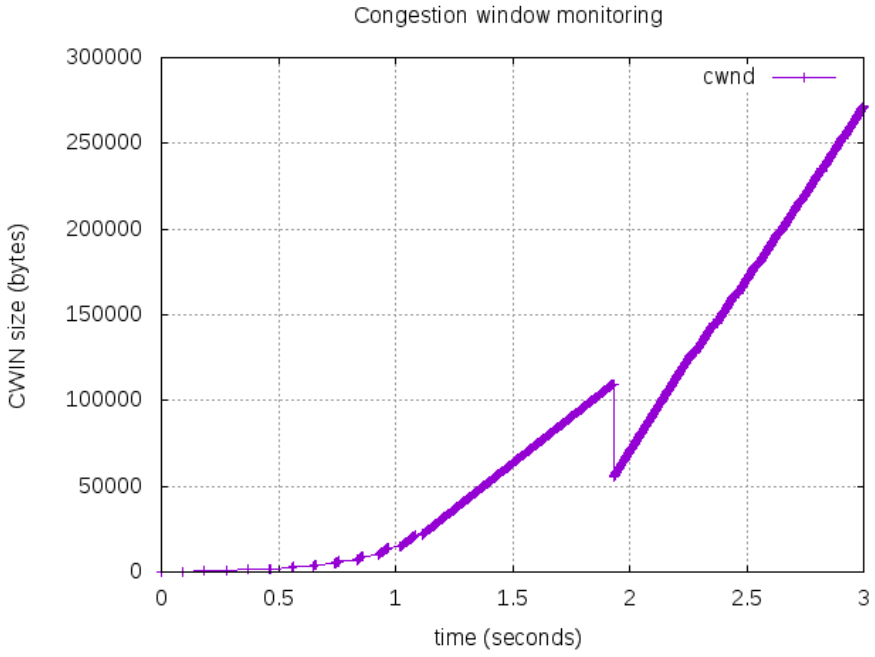

Figure 2: Reno CWND time 0.0s to 3.0s

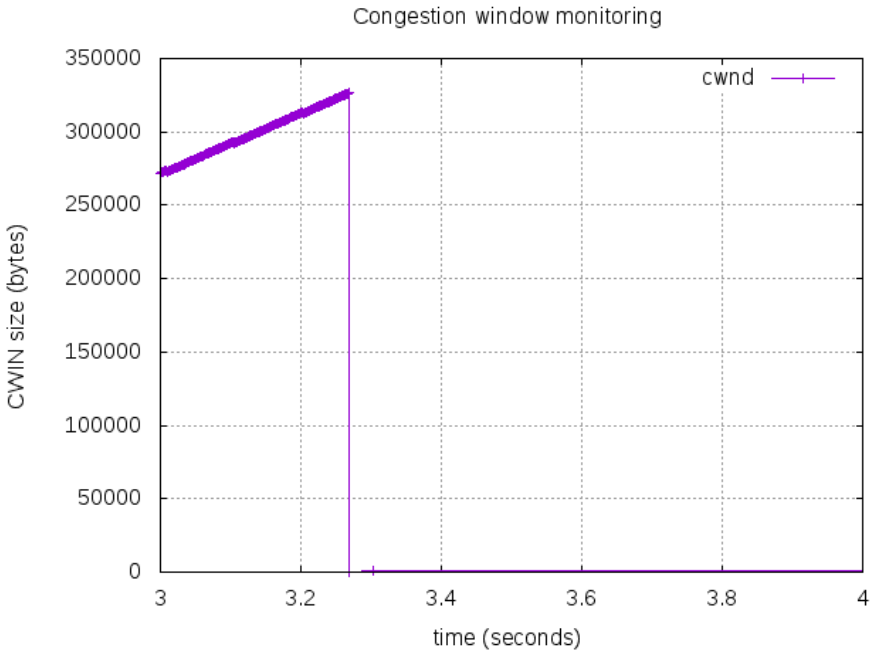

Figure 3: Reno CWND time 3.0s to 4.0s

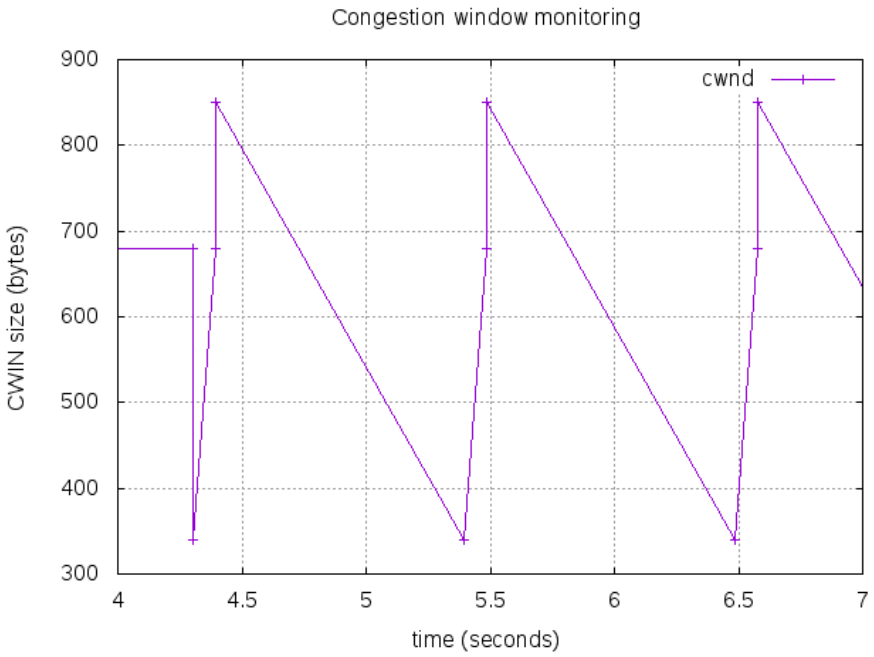

Figure 4: Reno CWND time 4.0s to 7.0s

Q1.2 Plot a graph showing SSTH versus time from 0.0s to 100.0s.

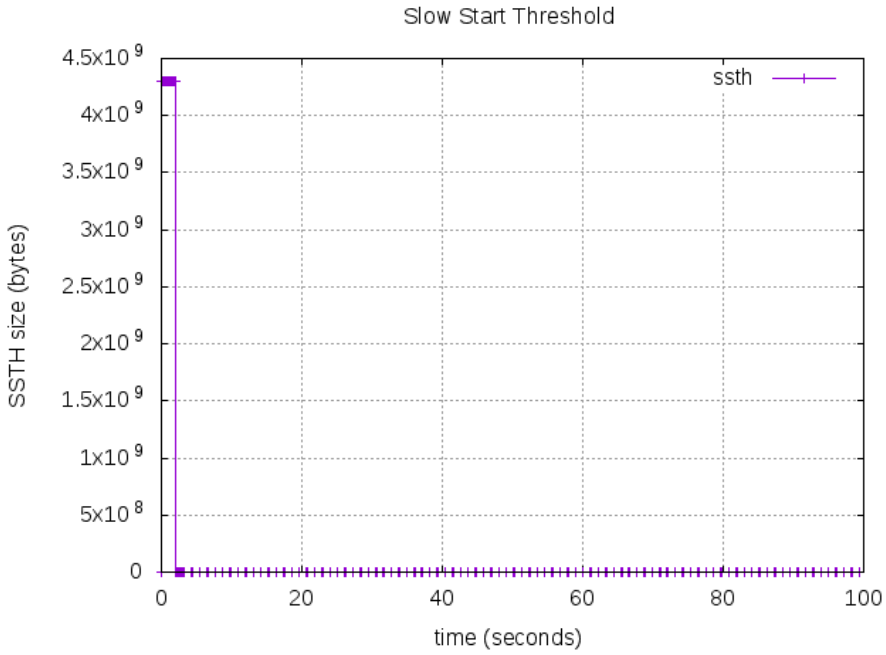

Figure 5: Reno SSTH time 0.0s to 100.0s

Q1.3 Find the points where the slow-start, congestion-avoidance, fast retransmit/fast recovery states begin.

Time (s)	Current CWND (bytes)	New CWND (bytes)	New State	Event
0.0905768	0	340	slow-start	connection initialization
1.93189	109480	55590	fast-recovery	duplicate ACK
3.26916	326570	340	slow-start	timeout
4.39503	680	850	congestion avoidance	$cwnd \geq ssht$

Q1.4 Plot a graph showing CWND versus time from 0.0s to 100.0s.

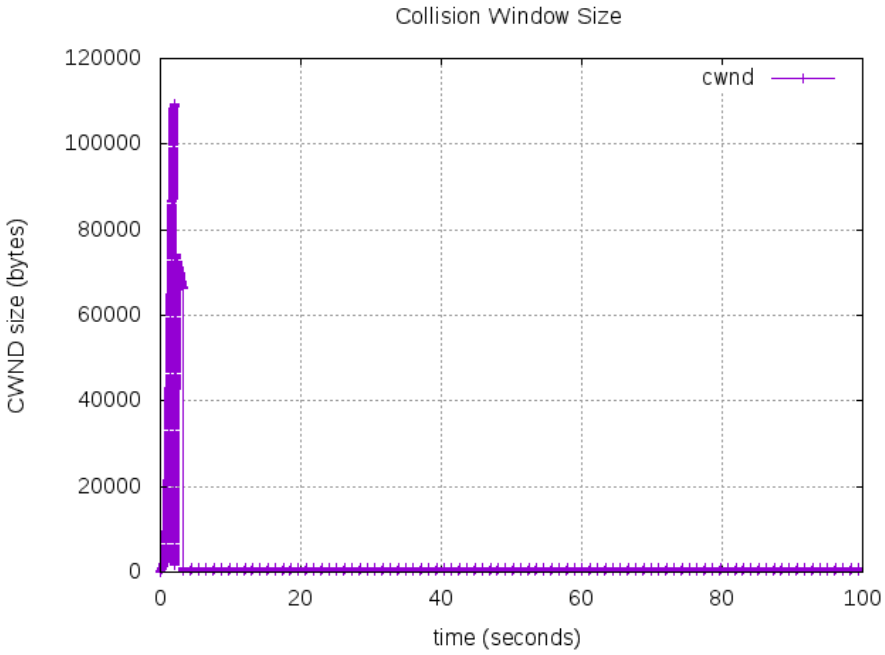

Figure 6: Westwood CWND time 0.0s to 100.0s

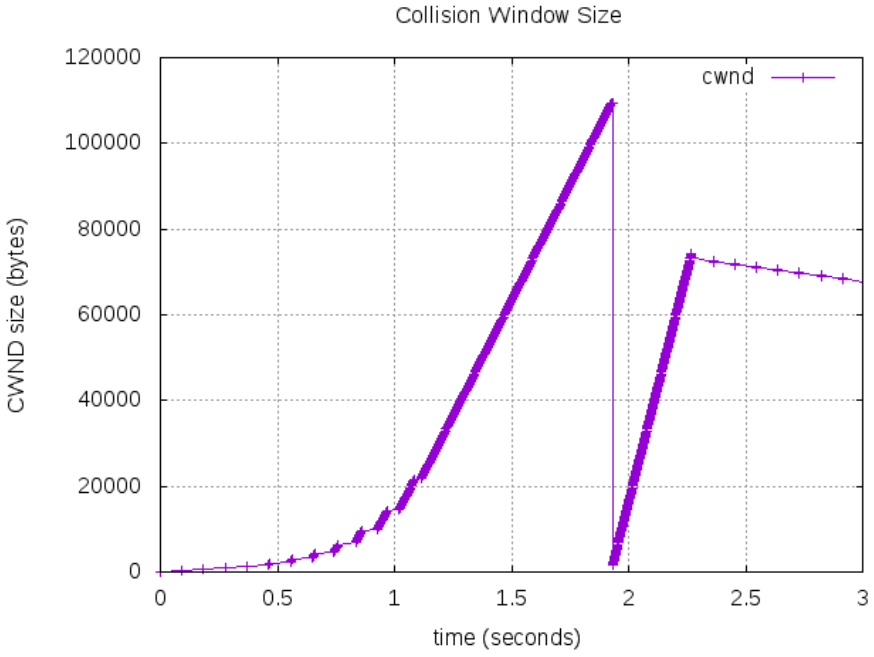

Figure 7: Westwood CWND time 0.0s to 3.0s

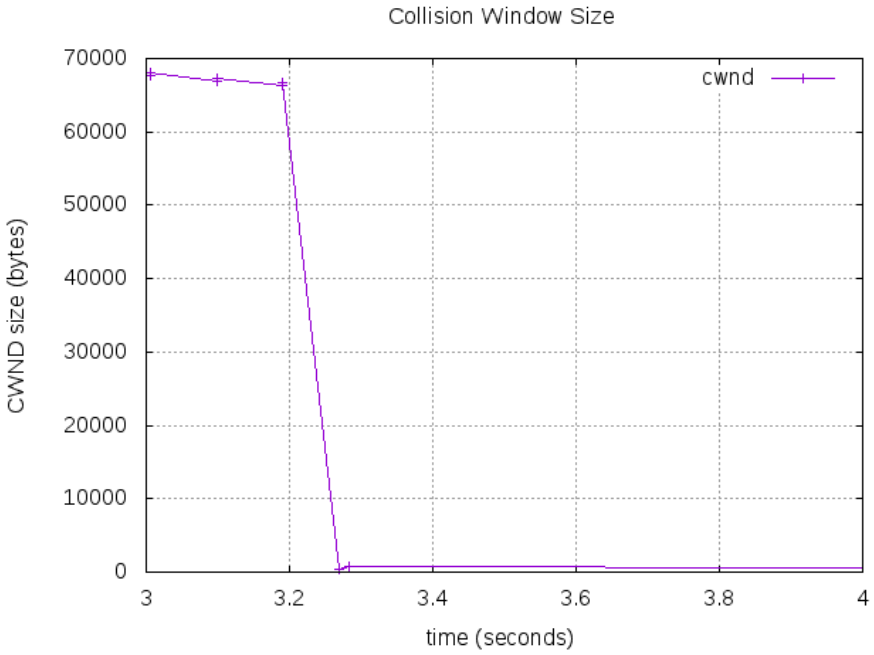

Figure 8: Westwood CWND time 3.0s to 4.0s

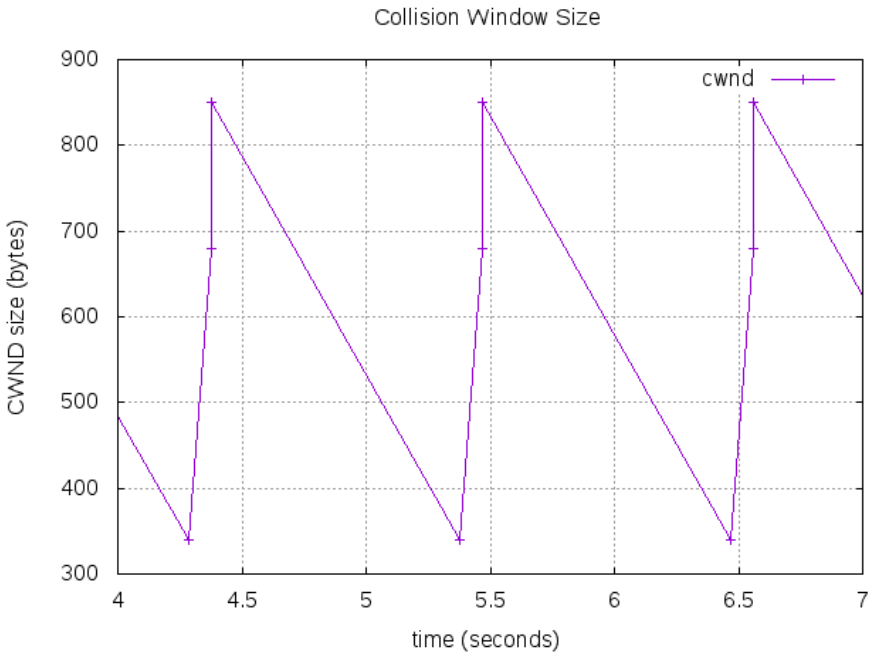

Figure 9: Westwood CWND time 4.0s to 7.0s

Q1.5 Plot a graph showing SSTH versus time from 0.0s to 100.0s.

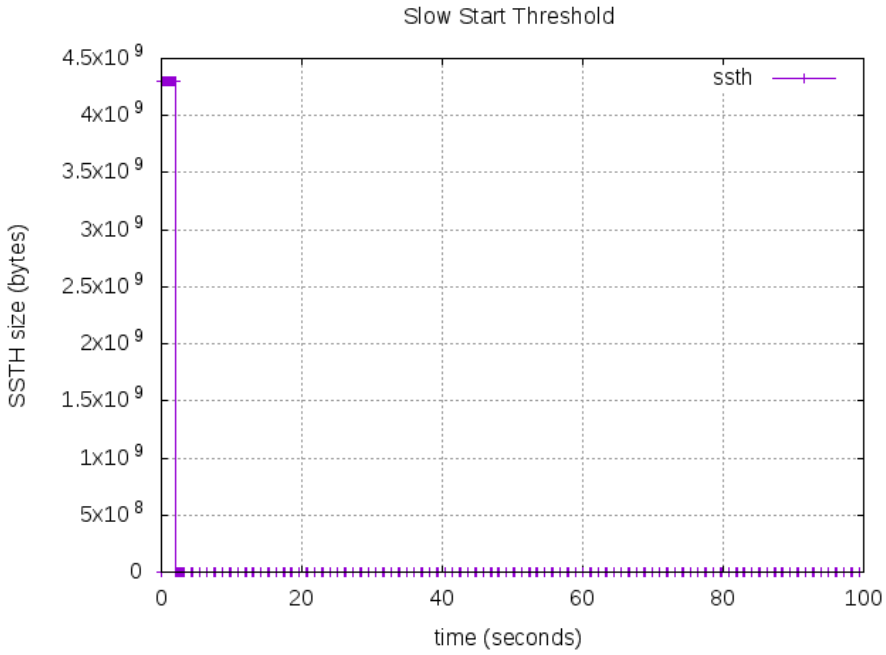

Figure 10: Westwood SSTH time 0.0s to 100.0s

Q1.6 Find the points where the slow-start, congestion-avoidance, fast retransmit/fast recovery states begin.

Time (s)	Current CWND (bytes)	New CWND (bytes)	New State	Event
0.0905768	0	340	slow-start	connection initialization
1.93189	109480	1700	congestion avoidance	duplicate ACK
3.19086	66640	340	slow-start	timeout
4.3752	680	850	congestion avoidance	$cwnd \geq ssht$

Q1.7 Discuss and motivate the differences you observe between the NewReno and this algorithm.

NewReno : Limits unknown packets being received. Limits the congestion window, and reset itself to a slow-start state. but if 3 of the same packets are received, it will halve the window, instead of reducing it to one. It changes the slow start threshold equal to that of the congestion window.
The algorithm that we used Westwood: A newer version of Reno, and another commonly used one. It controls parameters better, helping out streaming and overall quality of browsing the internet. One of the most 'fair' algorithms out there, and is one of the most efficient algorithms to date.

The state machines also seem to differ because New Reno stays in Slow start. However, Westwood has an extra set of slow start and congestion avoidance after the first drop.

Q2.1 Plot a graph showing the CWND and ssthresh versus time with all the data you get. These two metrics are in one graph.

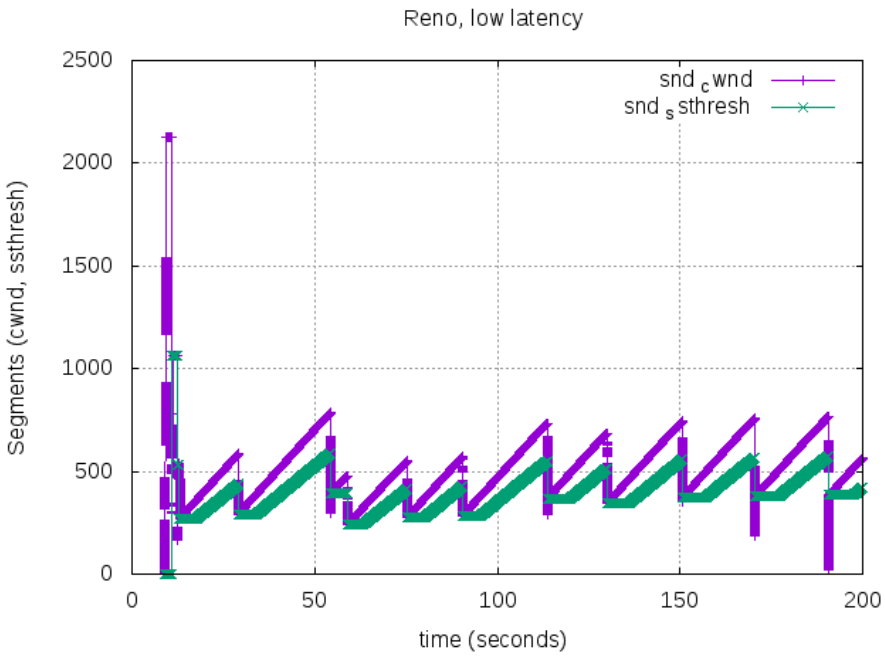

Figure 11: Reno Low Latency CWND and ssthresh versus time

Q2.2 Briefly discuss the changing process.

The starting point for CWND and SSTRESH are in default value. However, after the slow start phase the protocol enters in the congestion avoidance which makes the window to slowly increase until it gets multiple duplicate ACKs which triggers the fast recovery phase. During this transition the SSTRESH is cut in half and the congestion window is set to the new SSTRESH plus 3 MSS. The protocol transitions back to congestion avoidance and the congestion window slowly increases until it receives multiple duplicate ACKs after a new ACK is received. This will keep happening and repeating until the end of the experiment.

Q2.3 Plot a graph showing CWND versus time with all the data you get.

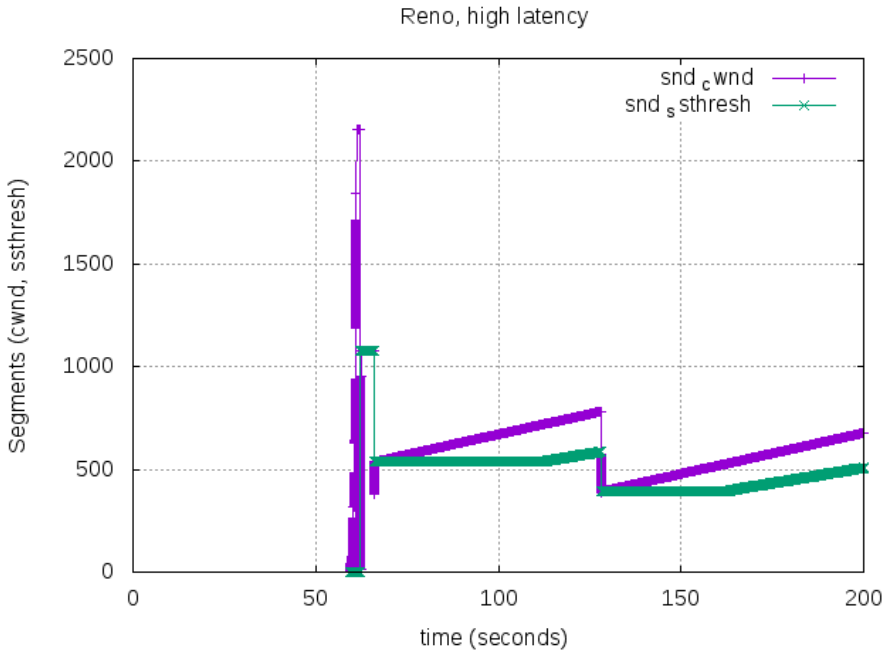

Figure 12: Reno High Latency CWND and ssthresh versus time

Q2.4 Compare this graph with the one from Q2.1, show the difference between these two graphs.

These two graphs both display the same pattern, although in Q2.3 the pattern much longer than in Q1.1. That happened because of the high latency. It takes longer for the source to be informed of packet loss at the sink.

Q2.5 Plot a graph showing CWND and ssthresh versus time with all the data you get.

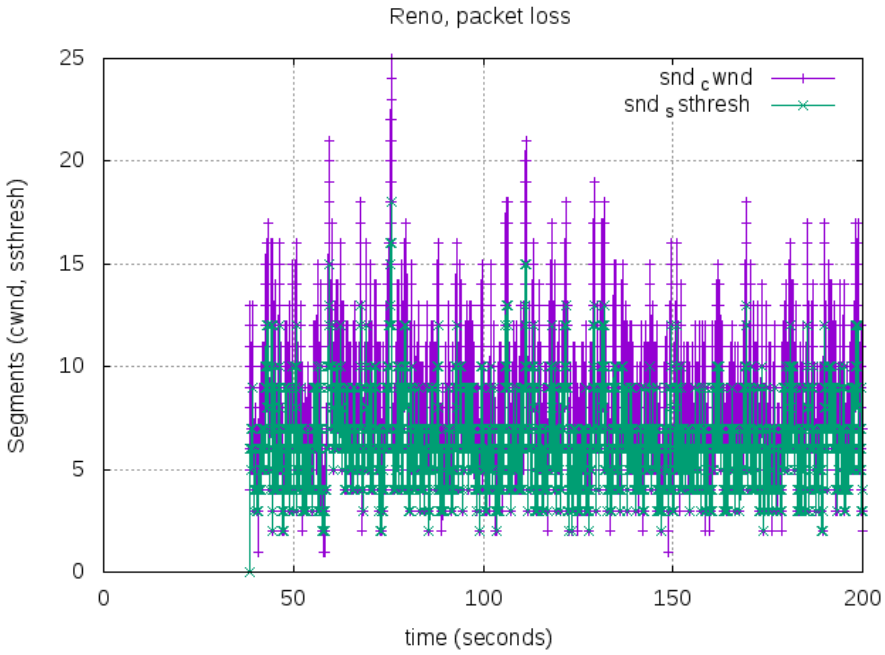

Figure 13: Reno Packet Loss CWND and ssthresh versus time

Q2.6 Compare this graph with the graph of Q2.1 discuss and point out the major differences.

Due to the 3% packet loss, duplicate ACKs are observed. Reno interprets it as congestion and adapts the CWND and SSTRESH accordingly and more frequently, resulting in a high pattern variations in the graph.

Q2.7 Zoom in the graph of this scenario (plot some parts of this scenario in a short duration, 10 or 20 seconds). Briefly explain the changing process.

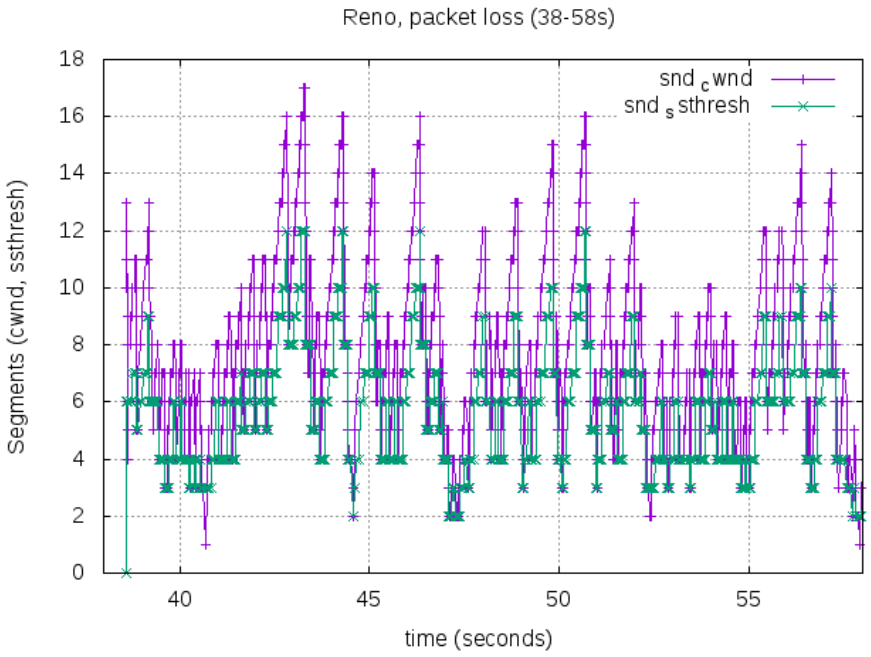

Figure 14: Reno Packet Loss CWND and ssthresh versus time (zoomed in)

In this graph (zoomed 38-58 seconds) we can notice that The SSTRESH can be seen to be constantly readjusted, and due to the 3% packet loss, TCP protocol appears to constantly shifting between phases.

Q2.8 Show a screen capture of the real throughput in this scenario.

```
vagrant@source:~$ iperf -c sink -Z reno -t 200
------------------------------------------------------------
Client connecting to sink, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.99.11 port 36130 connected with 192.168.99.12 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-201.3 sec  43.6 MBytes  1.82 Mbits/sec
```

Figure 15: Reno Packet Loss CWND and ssthresh screen capture

Q2.9 Plot a graph showing CWND and ssthresh versus time with all the data you get.

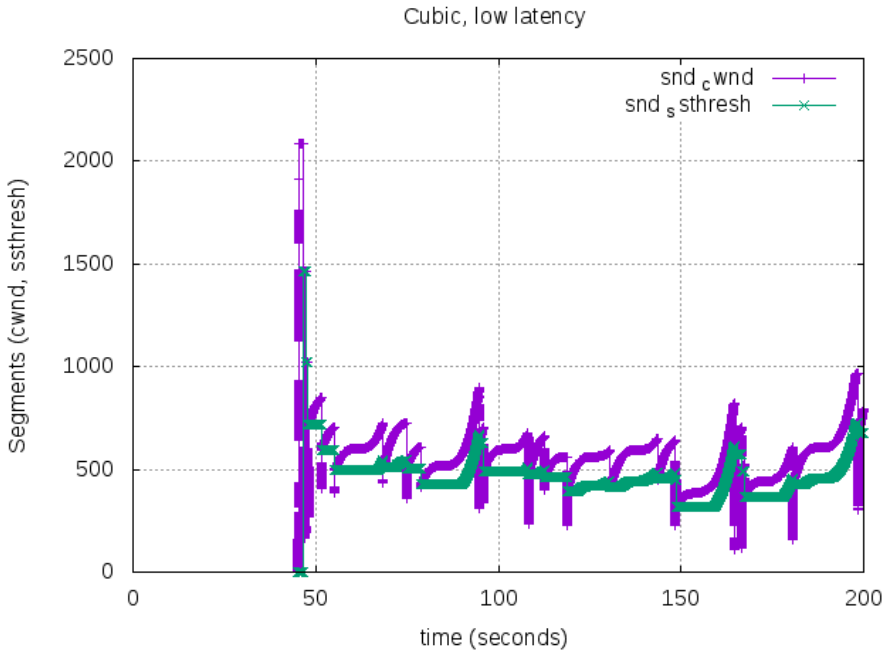

Figure 16: Cubic Low Latency CWND and ssthresh versus time

Q2.10 Compare this graph with the graph of Q2.1 discuss and point out the major differences.

The congestion window of Reno is calculated differently from Cubic, the latter sets the multiplicative window decrease to 0.7, while Reno takes half. The result is the CWND grows in a cubic function fashion and that can be observed on this graph.

Q2.11 Plot a graph showing CWND and ssthresh versus time with all the data you get.

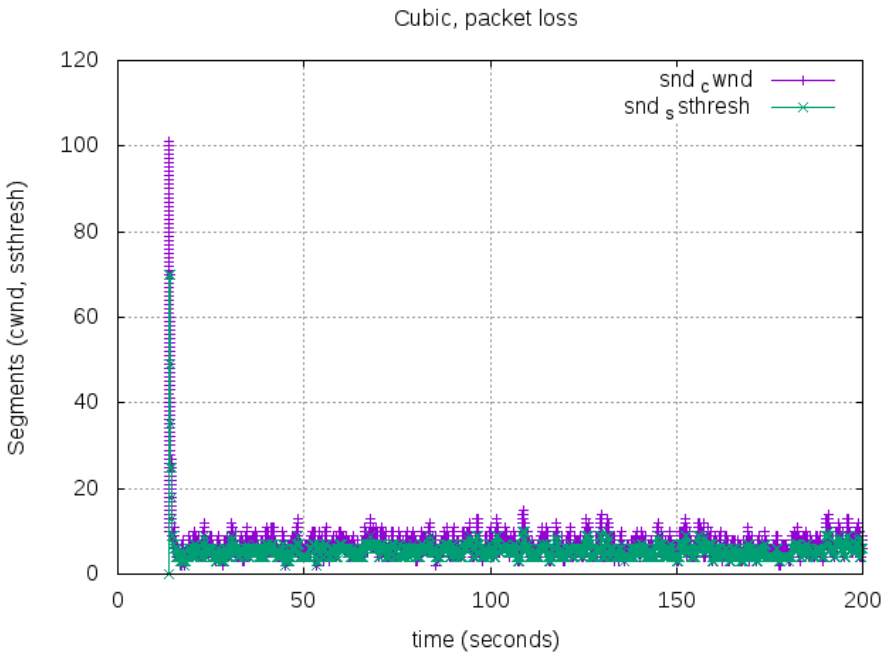

Figure 17: Cubic Packet Loss CWND and ssthresh versus time

Q2.12 Compare this graph with the graph of scenario three and show the differences.

We can clearly noticed that in scenario 3, the graph appears very crowded. However, The peaks points of the graphs are not high as in scenario 3, a possible explanation for that is that the overall throughput is lower.

Q2.13 Zoom in the graph of this scenario (plot some parts of this scenario in a short duration, 10 or 20 seconds). Briefly explain the changing process and compare it with the graph of Q2.7.

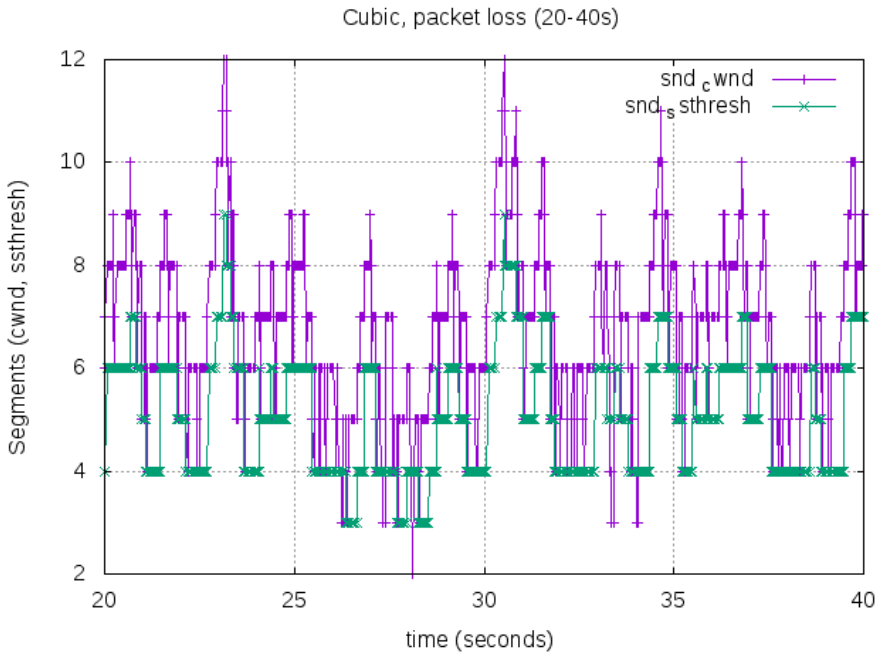

Figure 18: Cubic Packet Loss CWND and ssthresh versus time (zoomed in)

We derive from that Reno update its congestion window and slow start threshold more often. However, Cubic in the same time frame has fewer changes in the congestion window.

Q2.14 Show a screen capture of the real throughput and compare it with throughput of Q2.8. Tell the differences.

```
vagrant@source:~$ iperf -c sink -Z cubic -t 200
------------------------------------------------------------
Client connecting to sink, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.99.11 port 36136 connected with 192.168.99.12 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-201.0 sec  37.1 MBytes  1.55 Mbits/sec
```

Figure 19: Cubic Packet Loss CWND and ssthresh screen capture

We can clearly notice by comparing these two screen shots that the throughput in case of Reno is higher than Cubic.

Q3.1 Explain what an LFN network is. Change the simulation parameters to your likings and demonstrate that TcpNewReno is not suitable for LFN networks.

First off, Bandwidth-delay product (BDP) is the product of a data links capacity (in bps) and its round-trip delay time (in seconds). The result is the maximum amount of data (bits or bytes) on the network at any given time, that is data that has been transmitted but not yet acknowledged. Described in RFC 1072, networks with a BDP larger than 12500 bytes are considered as Long Fat Networks (LFN).

Q3.2 Explain SACK does. Change the simulation parameters to your likings and demonstrate the performance improvement with SACK.

Described in RFC 2018, SACK stands for Selective Acknowledgement, a mechanism that can help overcome the problem of a data sender only being to learn about a single lost packet per round trip time. The SACK option is included in a TCP segment sent from the data receiver to the data sender and informs the latter of non-contiguous blocks of data that have been received and queued.

```
./waf --run tcp-variants-comparison --command-template="%s --tracing=1
--pcap_tracing=1 --sack=1 --duration=100"
```

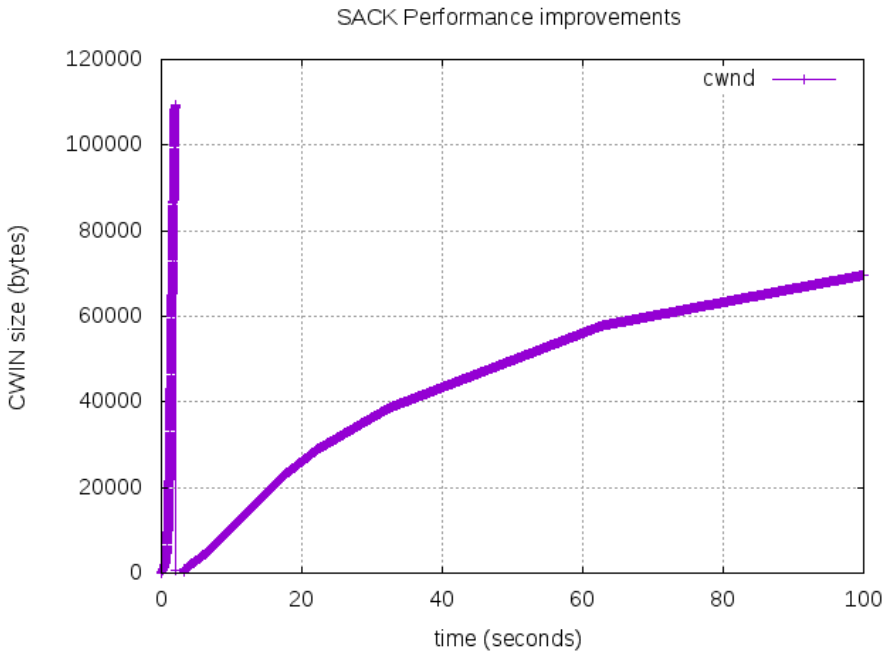

Figure 20: Selective Acknowledgment Performance improvement

Q3.3 Explain with TCP fairness is. Show the effect of multiple flows in the simulation.

TCP fairness is congestion control mechanisms for new network transmission protocols or peer-to-peer applications that must interact well with TCP. It requires that a new protocol receives no larger share of the network than a comparable TCP flow. This is important as TCP is the dominant transport protocol on the Internet, and if new protocols acquire unfair capacity they tend to cause problems such as congestion collapse. (Source: Wikipedia)

Q3.4 Replicate scenario 3 of the emulation: packet loss of 3%, delay of 50 ms and transfer duration of 200sec. Use TcpNewReno. Compare the two results.

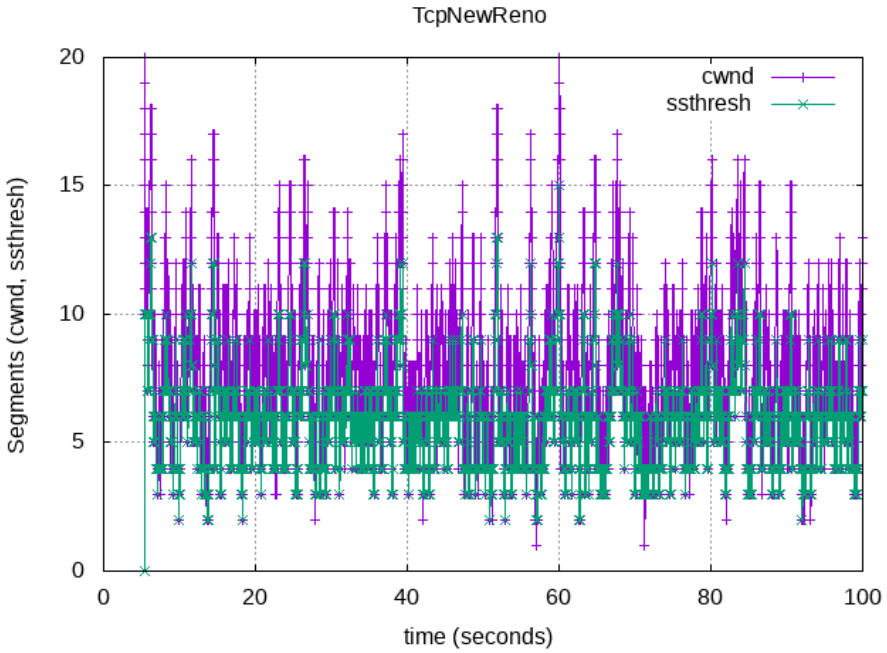

Figure 21: TCP New Reno with Packet Loss

As we see above the results are differences compared to scenario 3. This figure demonstrate the size of cwnd and sshthresh. However, scenario 3 shows the amount of segments which are sent. It turns to be more than the size of cwnd in bytes.

Q3.5 After these experiments, please briefly describe the difference between simulation and emulation?

An emulation emulates real working machines and runs the full software stack of the protocol in a real working environment and it runs in real time and takes into account all conditions, even those unintended. However, the simulation is a mathematical model intended to simulate conditions and apply the rules of the protocol to those conditions. It is more isolated and is more of a logical run of the protocol and it does not function as a real working environment and does not necessarily run in real time and .