

Alan Palayil (A20447935)

Professor: Yu Cheng

T.A.: Suyang Wang

Due Date: 4/30/2023

ECE 545 Project 2

In-Depth Investigation of TCP via NS3 Simulation

Contents

ECE 545 Project 2.....	1
I. Project Description:	3
II. Part 1: Efficiency of the TCP congestion control, flow control, and reliable data transfer protocols. .	3
Answers to Question 1:	4
Answers to Question 2:	6
Answers to Question 3:	8
III. Part 2: Resource sharing under the transport layer protocol	9
Answers to Question 1:	10
Answers to Question 2:	12
Answers to Question 3:	13
Answers to Question 4:	14
IV. Conclusion	16
V. Appendix.....	17
VI. Reference	18

I. Project Description:

This project focuses on studying the TCP protocol using ns3 simulation, which is a popular discrete event simulator for network analysis. The project aims to investigate different flavors of TCP and their behavior with varying background traffic. In addition, the project requires students to read related RFCs or other references to understand the implementation details of the TCP protocols considered in this project.

The network topology consists of six nodes and five links, with the bottleneck link being 3-4. All links have a propagation delay of 10ms and a drop-tail queue. The experiments start with default values for all variables, and then necessary modifications are made according to the problem specification. Each TCP connection is created with an FTP flow running on top of it, transferring data from the source node to the destination node. When the flavor of TCP is not mentioned, TCP Reno is assumed. The results and observations of the experiments must be supported by the simulation script and trace files, with graphs generated when necessary to better demonstrate the observations.

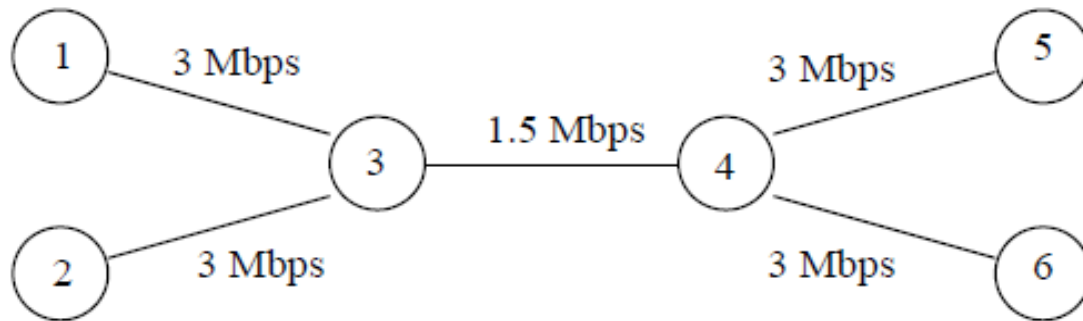


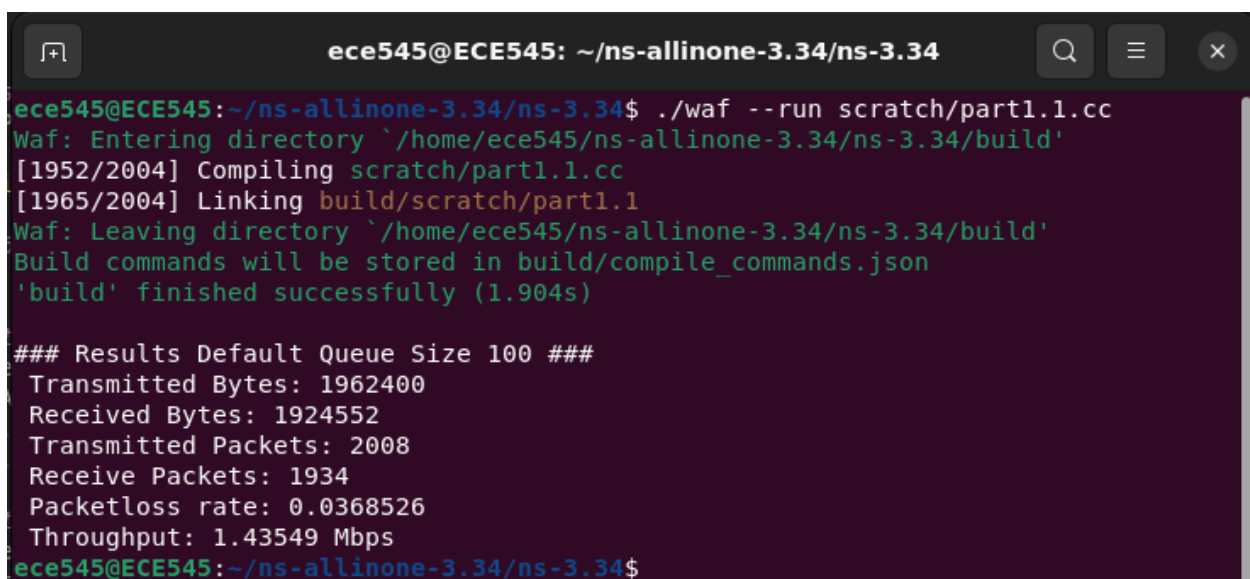
Figure 1: Network Topology

II. Part 1: Efficiency of the TCP congestion control, flow control, and reliable data transfer protocols.

In this part, the implementation of TCP protocol using ns3 simulation was studied. To create the simulation, the network topology with 6 nodes and 5 links was used, with the bottleneck located at link 3-4. The link capacities were given, and all links had a propagation delay of 10 ms and a drop-tail queue. The TCP connections were created with an FTP flow running on top of it, transferring data from the source node to the destination node. In this simulation, TCP Reno was assumed unless mentioned otherwise. The simulation parameters were set to default values with 0 to 10 seconds used to run the simulation.

To measure the round-trip time (RTT) in TCP and application, Rtttest.cc was used. For this part, the TCP connection was only considered between node 1 and node 5 as the source and destination, respectively. To create a bottleneck, a cwnd size was set for TCP Reno, allowing the modification to take place. The code was modelled based on Tcp-linux-reno.cc and a google link. In the simulation, a packet size of 1200 Byte/packet was used, and the sender's data rate was set to 10Mbps. The transmit and receive byte and packets, along with the packet loss rate and throughput, were shown in Figure 2. The simulation results were used to demonstrate the impact of the bottleneck and how it affects the TCP protocol behavior in the given network topology. Running the simulation with the default parameters from 0 to 10 seconds, we can answer Question 1.

Answers to Question 1:



```
ece545@ECE545: ~/ns-allinone-3.34/ns-3.34
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$ ./waf --run scratch/part1.1.cc
Waf: Entering directory `/home/ece545/ns-allinone-3.34/ns-3.34/build'
[1952/2004] Compiling scratch/part1.1.cc
[1965/2004] Linking build/scratch/part1.1
Waf: Leaving directory `/home/ece545/ns-allinone-3.34/ns-3.34/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.904s)

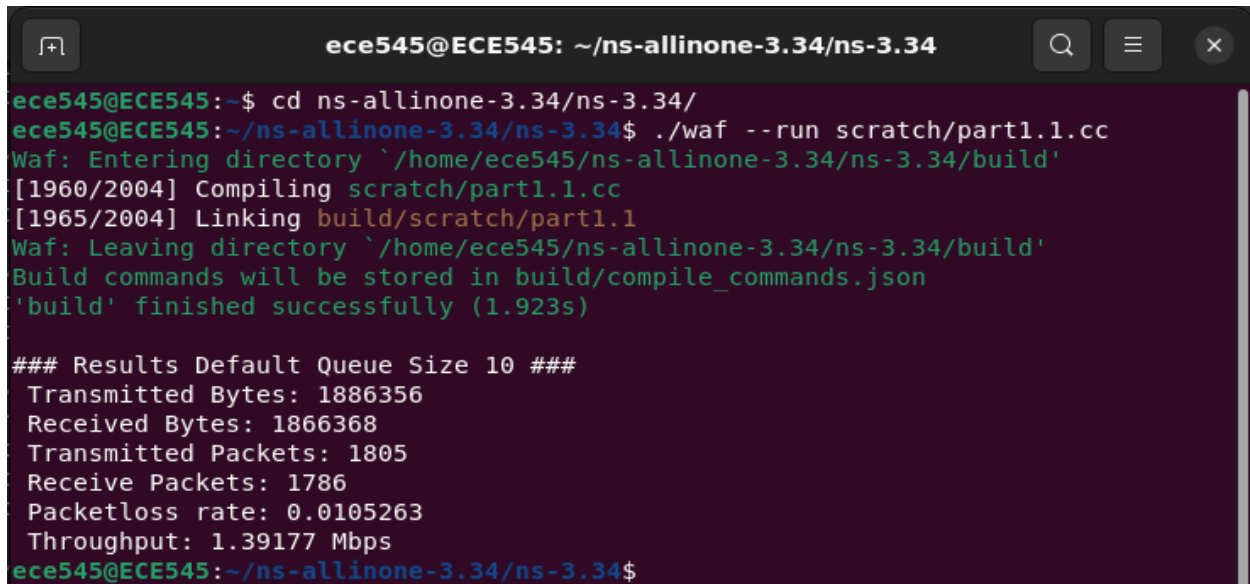
### Results Default Queue Size 100 ###
Transmitted Bytes: 1962400
Received Bytes: 1924552
Transmitted Packets: 2008
Receive Packets: 1934
Packetloss rate: 0.0368526
Throughput: 1.43549 Mbps
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$
```

Figure 2: Default Queue Size of 100.

- i. What is the total number of segments and the number of bytes successfully transmitted during the 10 seconds?
 - The number of successful transmission of bytes is 1962400 and packets is 2008.
- ii. What is the average throughput achieved? How does this compare with the bottleneck bandwidth in your topology?
 - Figure 2 illustrates that the throughput, constrained by the bottleneck bandwidth between node 3 and node 4, is 1.43549.
- iii. Then, change the queue size used at node 3 to a value much smaller than the default value in one case, and to a value much larger than the default value in the other case. Run simulations

to obtain the average throughput in these two cases, respectively. Do you get different average throughputs compared to that under the default queue size? Explain your observations?

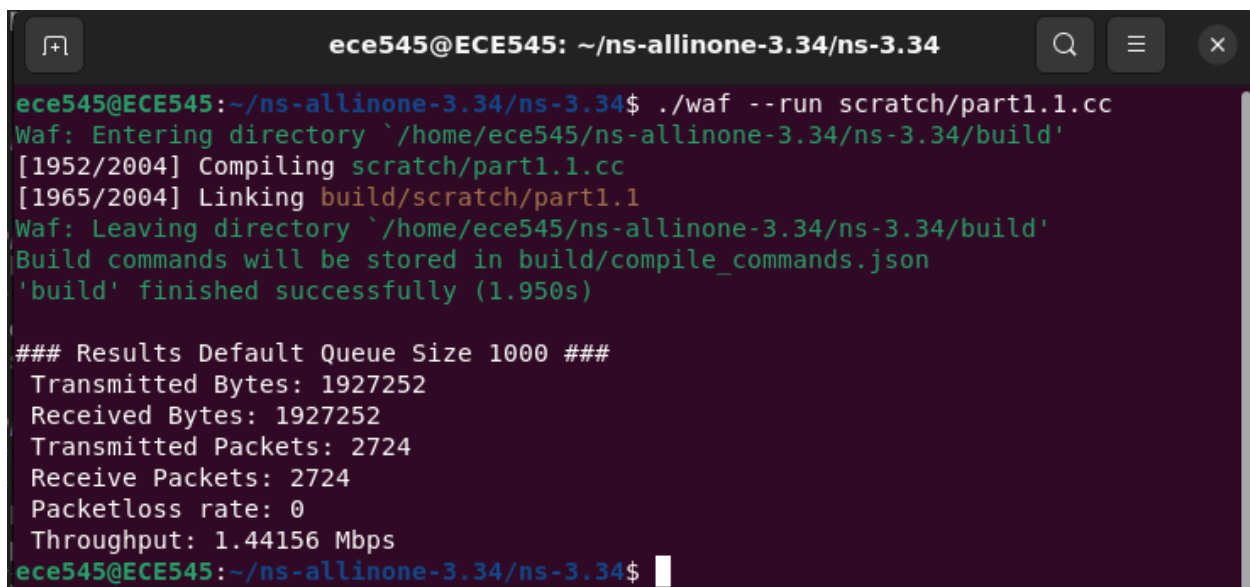
- The results for queue sizes of 10, 100, and 1000 are shown in Figures 2, 3, and 4, respectively. The simulation results indicate that with an extremely small queue size, the packet loss rate increases significantly. However, as the queue size increases to 100 and 1000, the results are similar to the default queue size.



```
ece545@ECE545: ~/ns-allinone-3.34/ns-3.34
ece545@ECE545:~$ cd ns-allinone-3.34/ns-3.34/
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$ ./waf --run scratch/part1.1.cc
Waf: Entering directory `/home/ece545/ns-allinone-3.34/ns-3.34/build'
[1960/2004] Compiling scratch/part1.1.cc
[1965/2004] Linking build/scratch/part1.1
Waf: Leaving directory `/home/ece545/ns-allinone-3.34/ns-3.34/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.923s)

### Results Default Queue Size 10 ###
Transmitted Bytes: 1886356
Received Bytes: 1866368
Transmitted Packets: 1805
Receive Packets: 1786
Packetloss rate: 0.0105263
Throughput: 1.39177 Mbps
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$
```

Figure 3: Queue Size of 10



```
ece545@ECE545: ~/ns-allinone-3.34/ns-3.34
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$ ./waf --run scratch/part1.1.cc
Waf: Entering directory `/home/ece545/ns-allinone-3.34/ns-3.34/build'
[1952/2004] Compiling scratch/part1.1.cc
[1965/2004] Linking build/scratch/part1.1
Waf: Leaving directory `/home/ece545/ns-allinone-3.34/ns-3.34/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.950s)

### Results Default Queue Size 1000 ###
Transmitted Bytes: 1927252
Received Bytes: 1927252
Transmitted Packets: 2724
Receive Packets: 2724
Packetloss rate: 0
Throughput: 1.44156 Mbps
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$
```

Figure 4: Queue Size of 1000

Answers to Question 2:

In order to observe the effects of packet dropping and timeout on retransmission and congestion control, it is recommended to set the queue size in the bottleneck link to a finite value that is appropriate. It is also important to set the receive window to a size that is large enough to bypass flow control. To achieve this, it was necessary to write appropriate code that can trace the congestion window, RTT, EstimatedRTT and TimeoutInterval. The simulation ran for a sufficiently long period of time in order to capture packet loss events and timeout events. Multiple simulation runs were done to capture such events if necessary.

- i. Plot the congestion window (cwnd) as a function of time. Use the graphs generated from one or multiple simulation runs to demonstrate the options of slow start, congestion avoidance, the reaction to triple duplicate ACK, and the reaction to timeout.
 - The sample ran for 100s seconds where it captured no congestion.

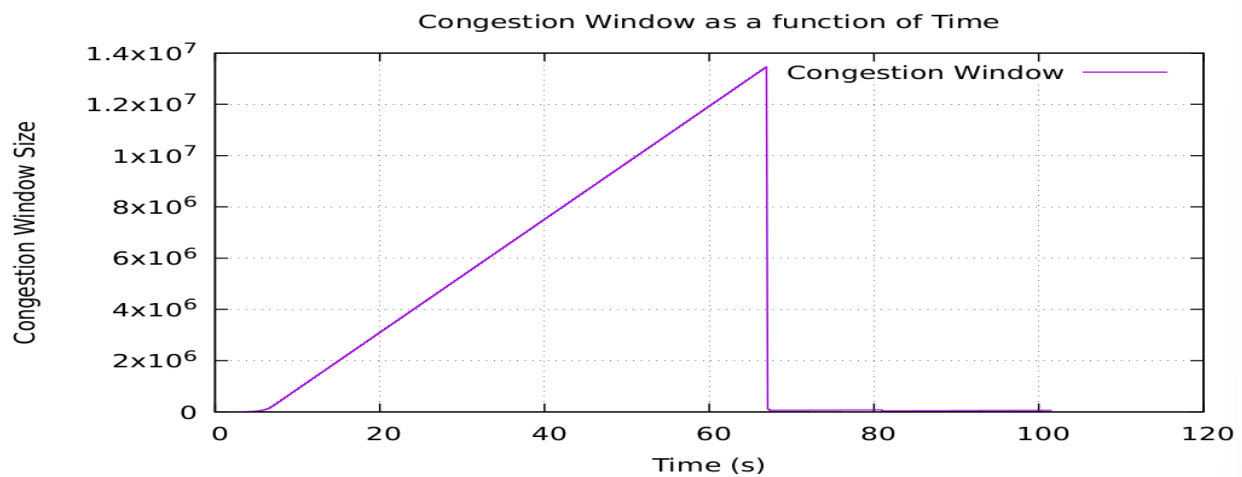


Figure 5: Congestion Window Size vs Time

- ii. Plot the RTT and EstimatedRTT as a function of time according to one of your trace files, to obtain a similar graph to Fig. 3.32 in our textbook.
 - The sample ran for 60 seconds.

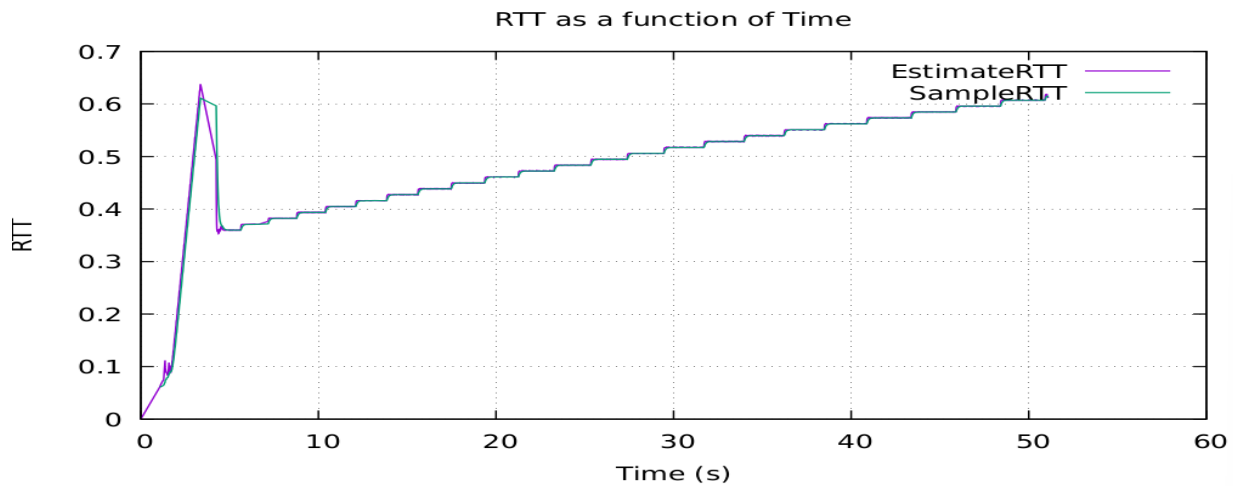


Figure 6: SampleRTT & EstimatedRTT Size vs Time

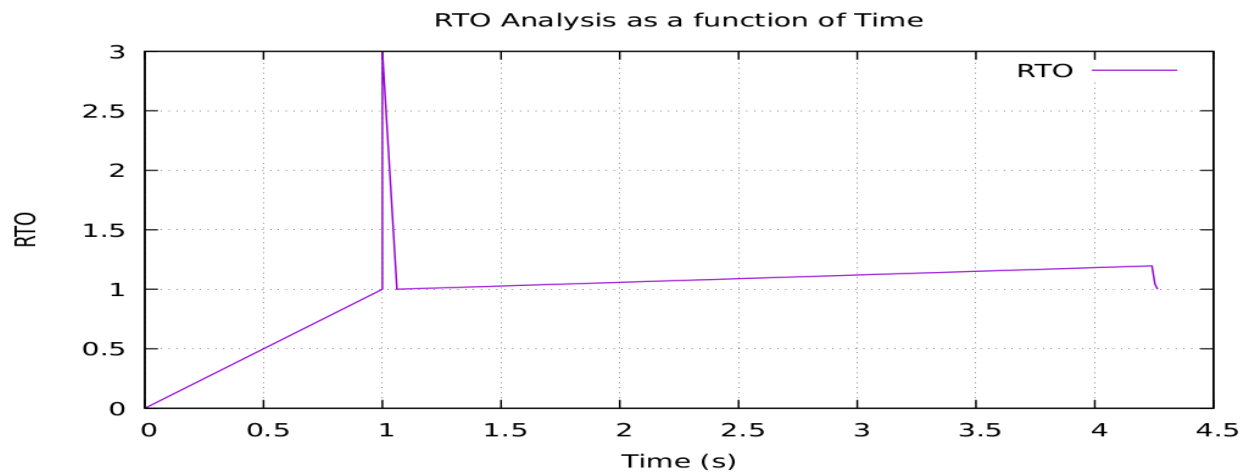


Figure 7: RTO Trace Analysis

- iii. Run simulations to investigate the relationship between the TCP throughput and the packet loss rate.

```
### Results Error rate 0.1 ###
Transmitted Bytes: 263216
Received Bytes: 238964
Transmitted Packets: 254
Receive Packets: 230
Packetloss rate: 0.0944882
Throughput: 0.0189934 Mbps
ece545@ECE545:~/ns-allinone-3.34/
```

Figure 8: Error Rate of 0.1

```
### Results Error rate 0.01 ###
Transmitted Bytes: 1968452
Received Bytes: 1944256
Transmitted Packets: 1874
Receive Packets: 1851
Packetloss rate: 0.0122732
Throughput: 0.154901 Mbps
ece545@ECE545:~/ns-allinone-3.34/
```

Figure 9: Error Rate of 0.01

```

### Results Error rate 0.001 ###
Transmitted Bytes: 8781760
Received Bytes: 8770188
Transmitted Packets: 8903
Receive Packets: 8892
Packetloss rate: 0.00123554
Throughput: 0.696178 Mbps
ece545@ECE545:~/ns-allinone-3.34/r

```

Figure 10: Error Rate of 0.001

```

### Results Error rate 0.0001 ###
Transmitted Bytes: 25475144
Received Bytes: 25473600
Transmitted Packets: 31920
Receive Packets: 31917
Packetloss rate: 0.00093985
Throughput: 2.02514 Mbps
ece545@ECE545:~/ns-allinone-3.34/r

```

Figure 11: Error Rate of 0.0001

Subsequent experiments (including Part 2). If necessary, the value of the receive window and other related parameters were adjusted to create a scenario that demonstrated the effect of flow control. Necessary trace data or graphs were used to illustrate the observations of the flow control option.

Answers to Question 3:

To demonstrate the effect of flow control, the receive window and related parameters were adjusted. The network's performance was observed with a receive window size of packet-size, packet-size*10, and packet-size*100, which are shown in Figures 12, 13, and 14, respectively. The results indicate that flow control can reduce the packet loss rate but may also decrease the throughput.

```

### Results Received Window Size = Packet-Size*0.1 ###
Transmitted Bytes: 160
Received Bytes: 160
Transmitted Packets: 3
Receive Packets: 3
Packetloss rate: 0
Throughput: 2.55844e-05 Mbps
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$

```

Figure 12: Received Window Size = Packet Size * 0.1

```

### Results Received Window Size = Packet-Size ###
Transmitted Bytes: 196884
Received Bytes: 196884
Transmitted Packets: 190
Receive Packets: 190
Packetloss rate: 0
Throughput: 0.0309089 Mbps
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$

```

Figure 13: Received Window Size = Packet Size


```

### Results Received Window Size = Packet-Size*10 ###
Transmitted Bytes: 9408688
Received Bytes: 9408688
Transmitted Packets: 8953
Receive Packets: 8953
Packetloss rate: 0
Throughput: 1.48394 Mbps
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$

```

Figure 14: Received Window Size = Packet Size * 10

```

### Results Received Window Size = Packet-Size*100 ###
Transmitted Bytes: 9453388
Received Bytes: 9410180
Transmitted Packets: 9097
Receive Packets: 9043
Packetloss rate: 0.00593602
Throughput: 1.48423 Mbps
ece545@ECE545:~/ns-allinone-3.34/ns-3.34$

```

Figure 15: Received Window Size = Packet Size * 100

III. Part 2: Resource sharing under the transport layer protocol

In the second part of the experiment, two transport-layer flows were created between different nodes. Flow-1 was established between node-1 as the source and node-5 as the destination, while Flow-2 was established between node-2 as the source and node-6 as the destination. The goal was to observe the resource sharing between these two flows. To achieve this, a combination of TCP and UDP was used, with Flow-1 using TCP while Flow-2 used UDP with a CBR source. The experiment was run multiple times with varying traffic generation rates for the UDP flow, and the throughput achieved by the two flows was observed. A graph was plotted, with the UDP throughput on the x-axis and the TCP throughput on the y-axis. The UDP throughput for which the two flows achieved a fair share of the link was determined and fixed. Further simulations were run to calculate the loss rate of the TCP connection by replacing the drop-tail queue with another queuing scheme provided by ns3. The simulation was run again to determine if there were any differences in the throughput achieved under the new queuing scheme. Possible reasons leading to the TCP throughput increase or decrease under a certain queuing scheme were investigated through extra reading.

Answers to Question 1:

- i. Plot a graph showing the UDP throughput on the x-axis and TCP throughput on the y-axis.
- I tested the CBR traffic generation rate at 0.5 Mbps (Figure 16), 0.725 Mbps (Figure 17), and 1 Mbps (Figure 18). Meanwhile, the TCP sending rate remained constant at 10 Mbps.

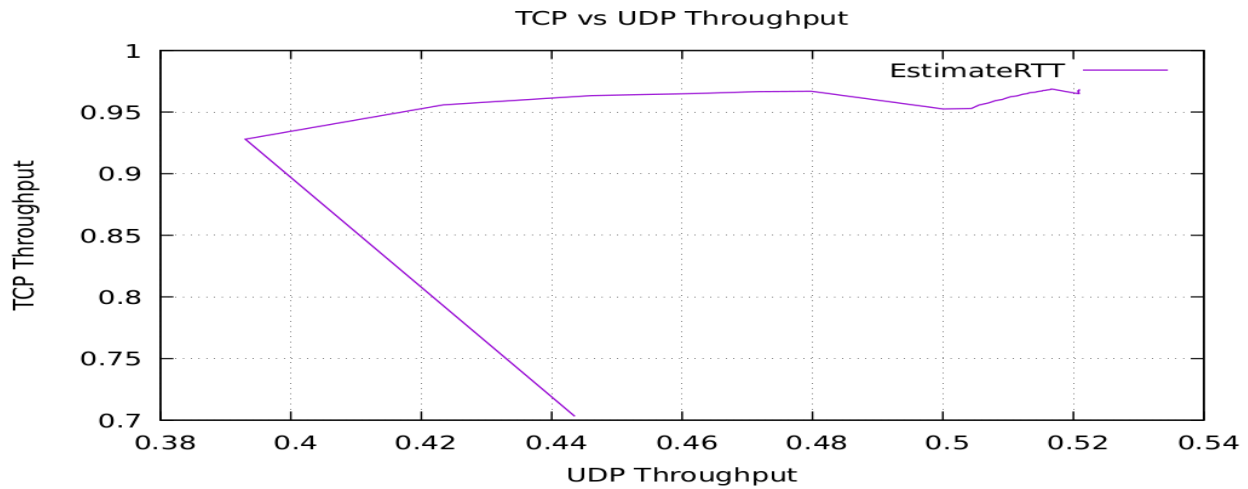


Figure 16: UDP vs TCP Throughput 0.5 Mbps

- ii. Determine the UDP throughput for which the two flows achieve a fair share of the link.
- To achieve a fair share of the link, the UDP data rate would be set to 0.725 Mbps. This means that the total number of packages passed would be roughly equal among all the different types of traffic.

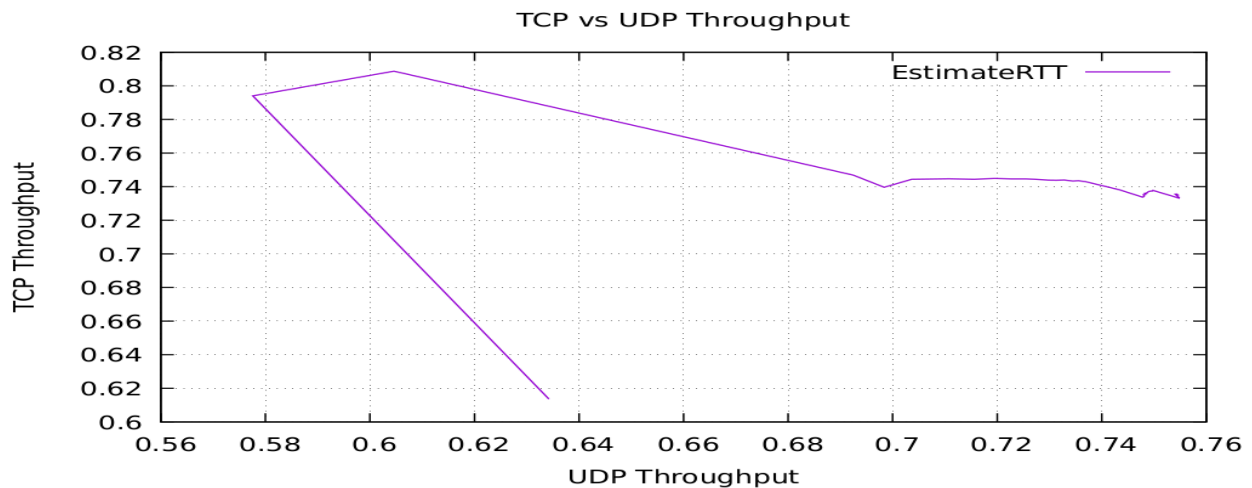


Figure 17: UDP vs TCP Throughput 0.725 Mbps

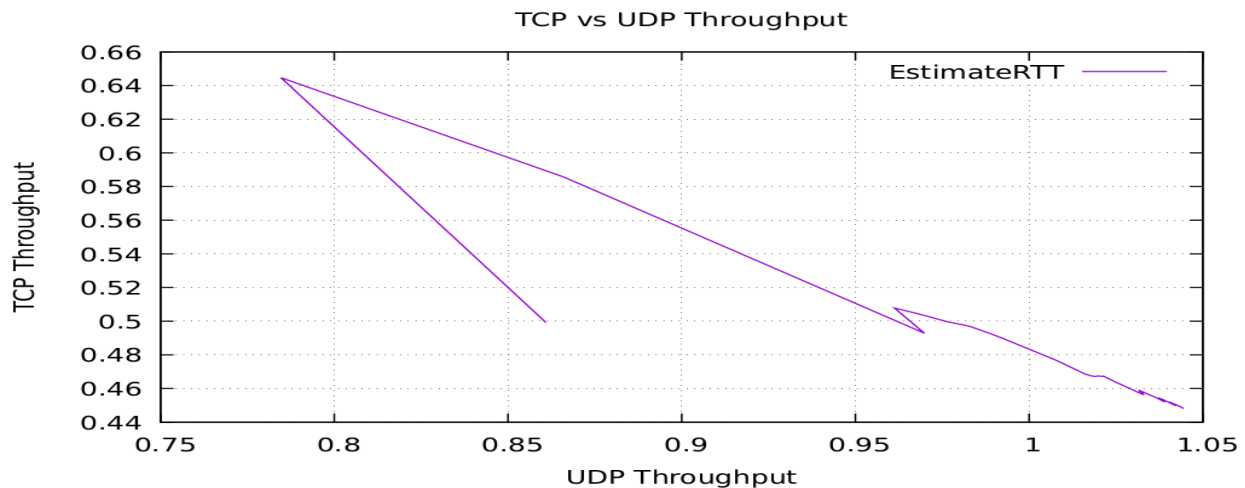


Figure 18: UDP vs TCP Throughput 1 Mbps

- iii. Fix the UDP rate at the fair-share rate you just found and run simulations to calculate the loss rate of the TCP connection.
 - The packet loss rate ranges from 0.00564972 to 0.00746269, which translates to approximately 0.5% to 0.7%.

```

### Results for TCP/UDP CBR ###
Flow 1
  Transmitted Bytes: 4769476
  Received Bytes: 4733708
  Transmitted Packets: 4556
  Receive Packets: 4522
  Packetloss rate: 0.00746269
  Throughput: 0.743181 Mbps
Flow 2
  Transmitted Bytes: 4779000
  Received Bytes: 4752000
  Transmitted Packets: 8850
  Receive Packets: 8800
  Packetloss rate: 0.00564972
  Throughput: 0.754782 Mbps
Flow 3
  Transmitted Bytes: 129432
  Received Bytes: 129432
  Transmitted Packets: 2393
  Receive Packets: 2393
  Packetloss rate: 0
  Throughput: 0.0203449 Mbps

```

Figure 19: Results using CBR.

- iv. Replace the drop-tail queue with another queuing scheme provided by ns3 and run simulation again. Do you get different throughput under a new queuing scheme?
- Based on the predetermined size of the queue, it can be inferred that the throughput will not significantly vary even in the absence of a drop queue mechanism.

Answers to Question 2:

The experiment involves two TCP flows, each with a different maximum window size (MWS) - 30 packets for flow-1 and 6 packets for flow-2, respectively. The throughput of both flows over time is plotted on a single graph. To avoid overloading the bottleneck link, the TCP data rate is set at 0.5, 1, and 1.5Mbps.

The results indicate that varying contention window sizes result in significantly different throughputs, leading to varying link fairness. This is primarily attributed to the frequency of congestion events and their impact on throughput. Higher instances of congestion lead to lower throughput. However, decreasing the TCP data rate shows a slight similarity in throughput over time despite the differing congestion window sizes.

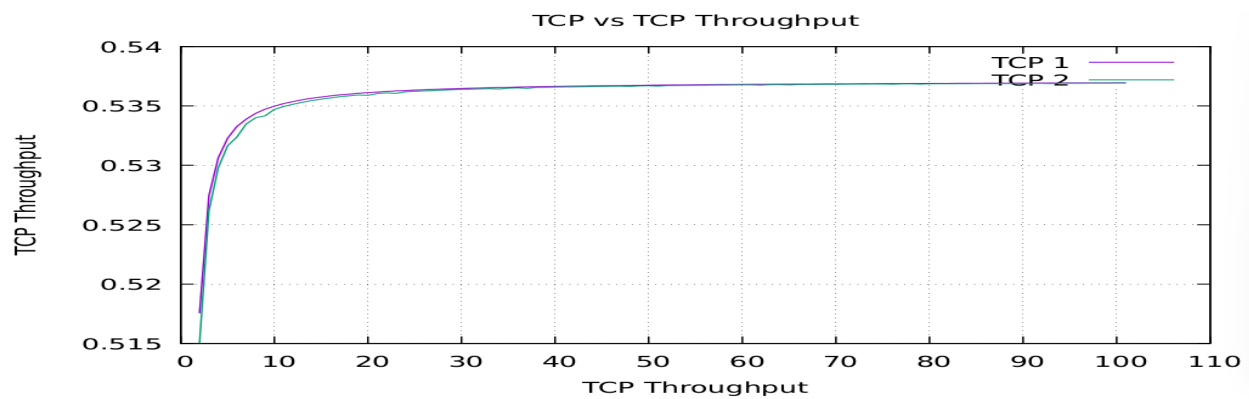


Figure 20: Data Rate of TCP is 0.5 Mbps

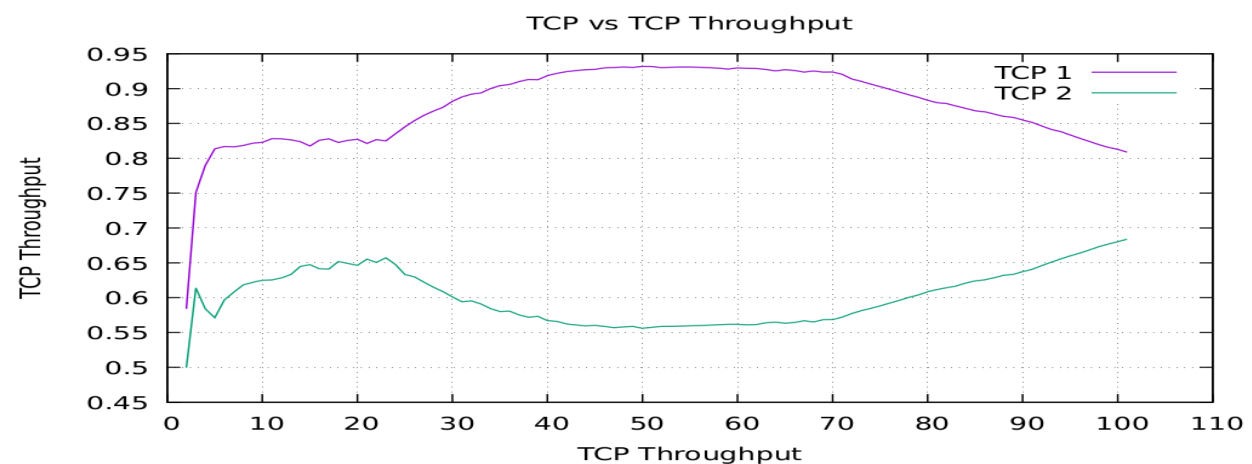


Figure 21: Data Rate of TCP is 1 Mbps

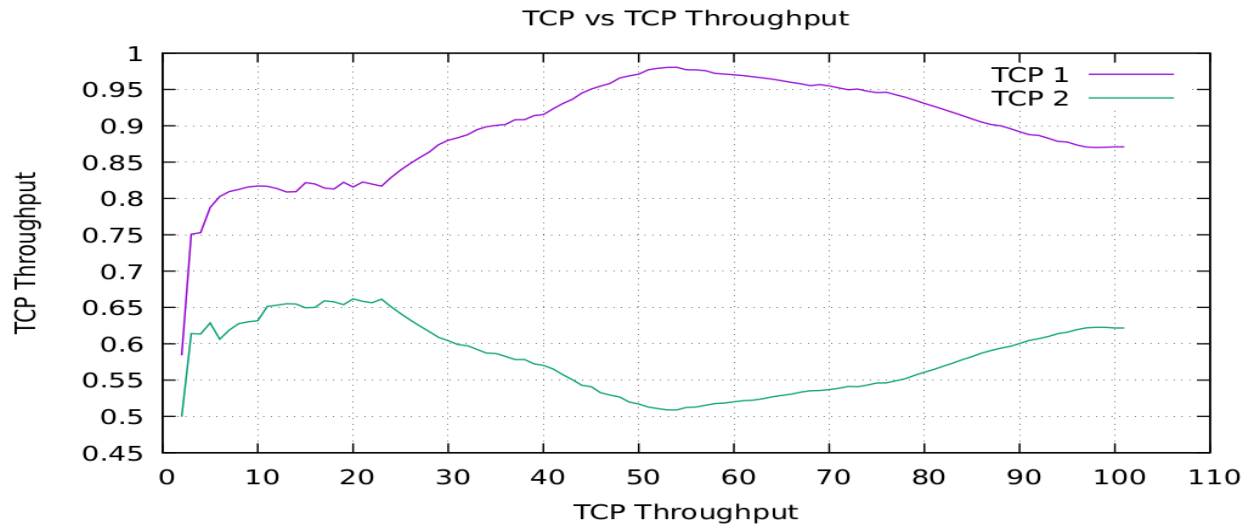


Figure 22: Data Rate of TCP is 1.5 Mbps

Answers to Question 3:

In this experiment, two TCP flows are employed with a packet size of 1000 bytes for flow-1 and 500 bytes for flow-2. To observe the impact of smaller packet sizes, the TCP transmission rate is reduced to 0.5Mbps. This enables a comparison to be made against when the link is operating at 1 or 1.5Mbps.

The results reveal that in a better network, a smaller packet size has a detrimental effect on throughput as it provides no advantage and increases the transmission time by two-fold, resulting in lower throughput. However, when the transmission data rate is poor, smaller packets have a higher likelihood of being transmitted during link congestion, resulting in higher throughput with smaller packet sizes.

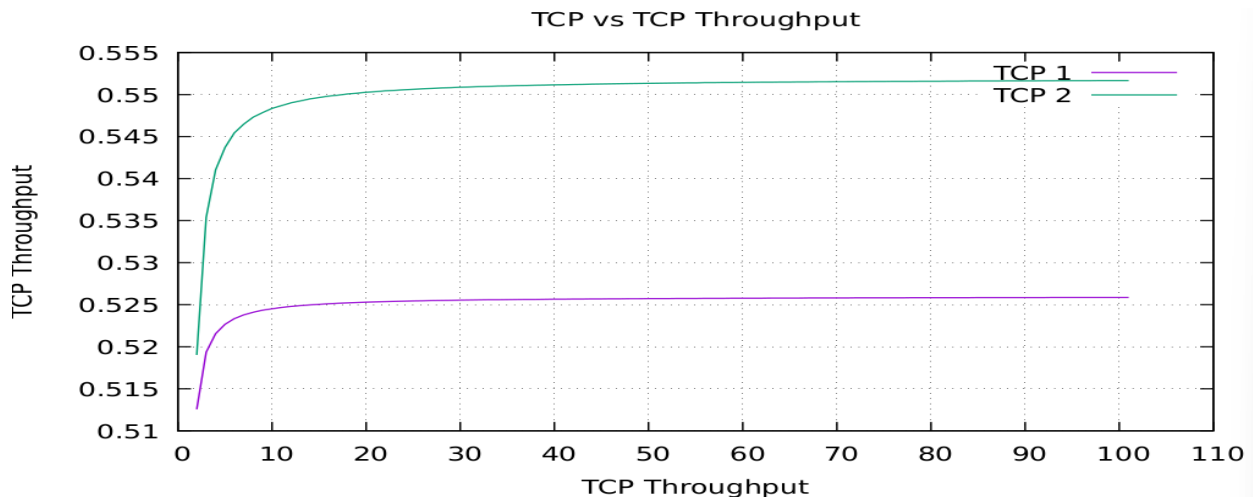


Figure 23: Data Rate of TCP is 0.5 Mbps

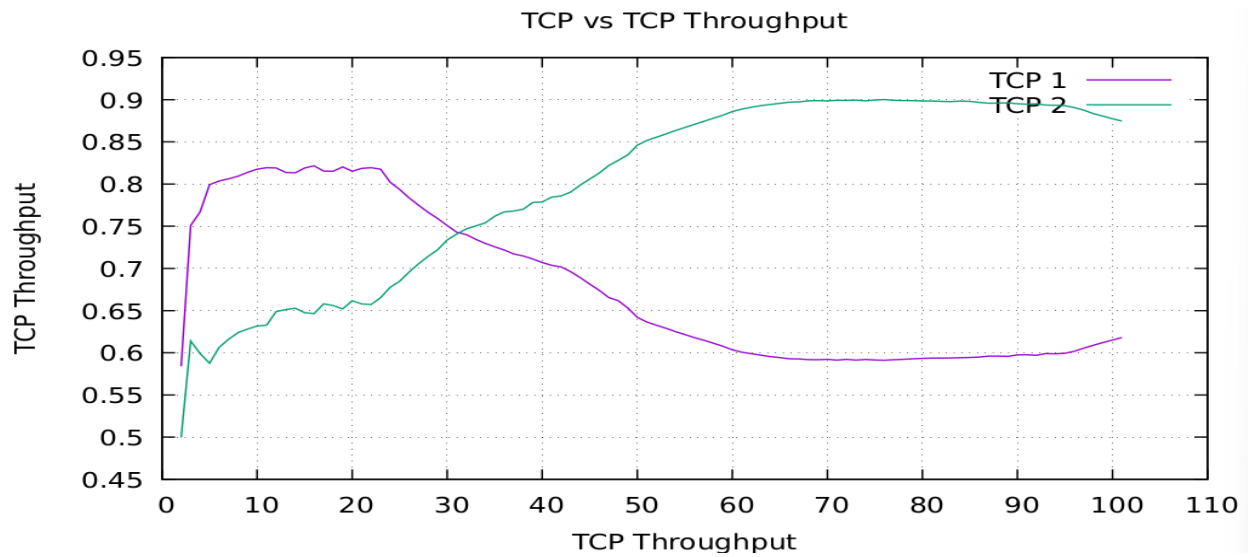


Figure 24: Data Rate of TCP is 1 Mbps

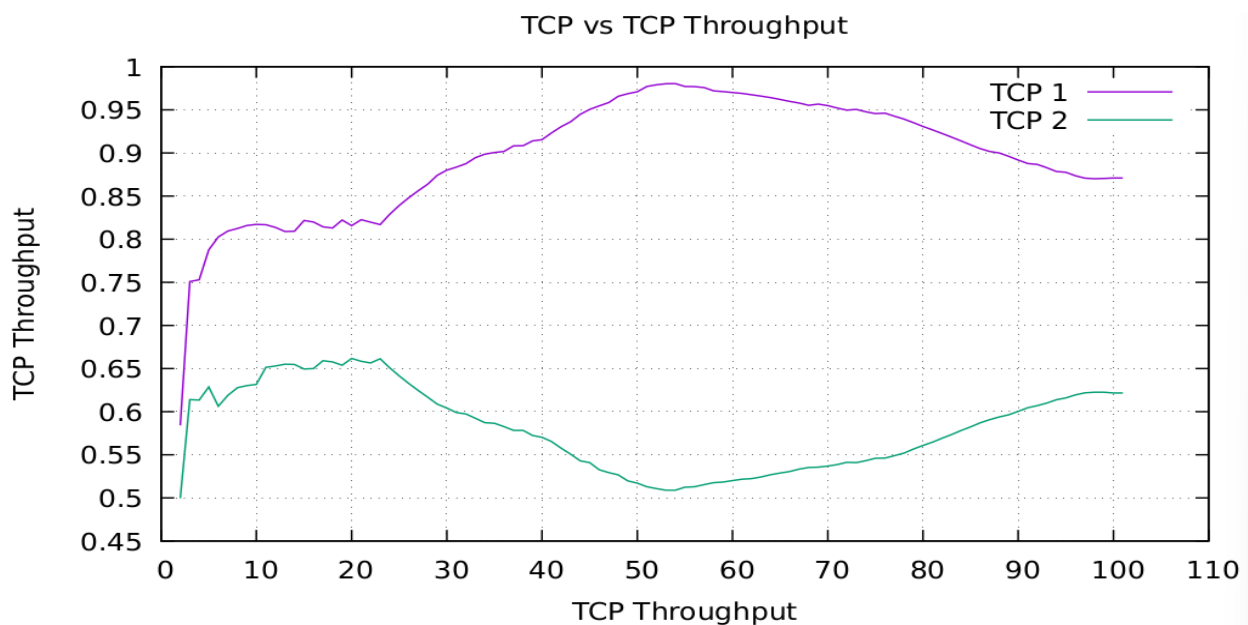


Figure 25: Data Rate of TCP is 1.5 Mbps

Answers to Question 4:

The experiment involves two TCP flows - flow-1 is TCP Reno and flow-2 is TCP Westwood - with the same maximum window size (MWS) selected for both flows. Analysis of the throughput vs. throughput figure reveals that TCP Reno and TCP Westwood obtained a reasonably equitable share of the link. However, further investigation using the time graph highlights that TCP Reno performs slightly worse than TCP Westwood over time, especially when the link data rate is adequate.

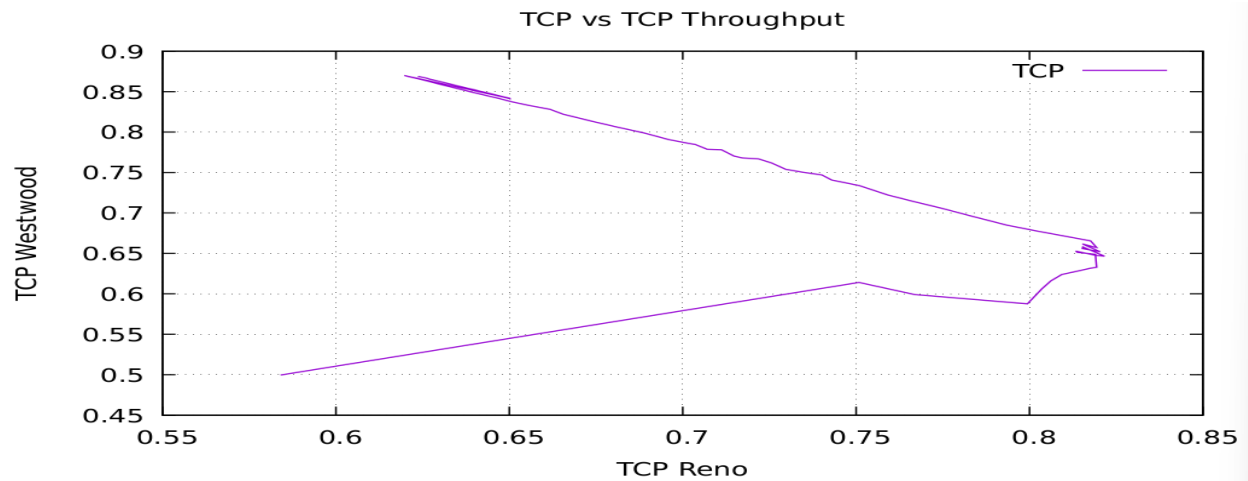


Figure 26: Reno vs Westwood without time and Data rate of 1 Mbps

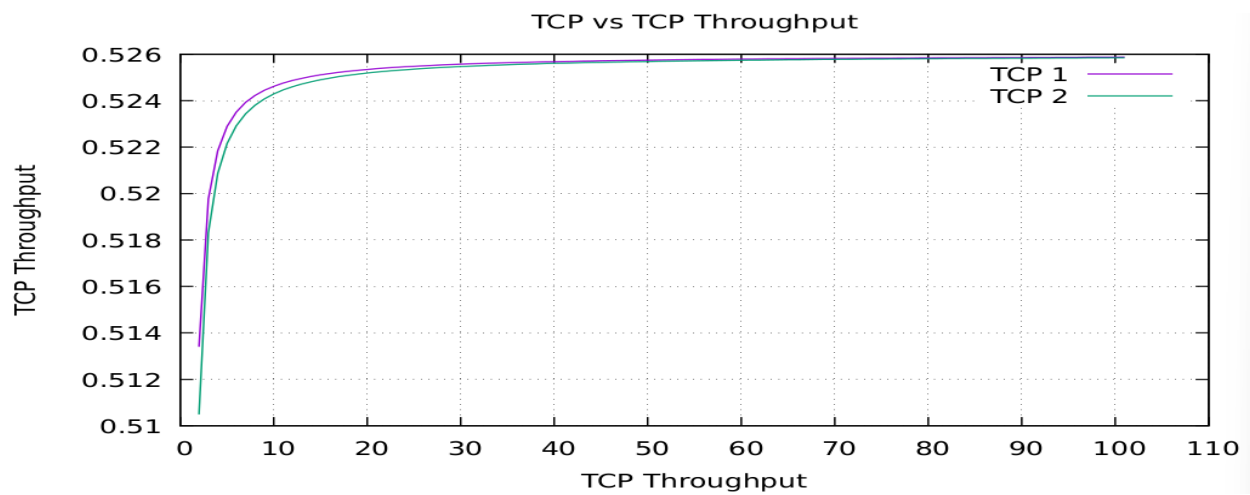


Figure 27: Reno vs Westwood with Data rate of 0.5 Mbps

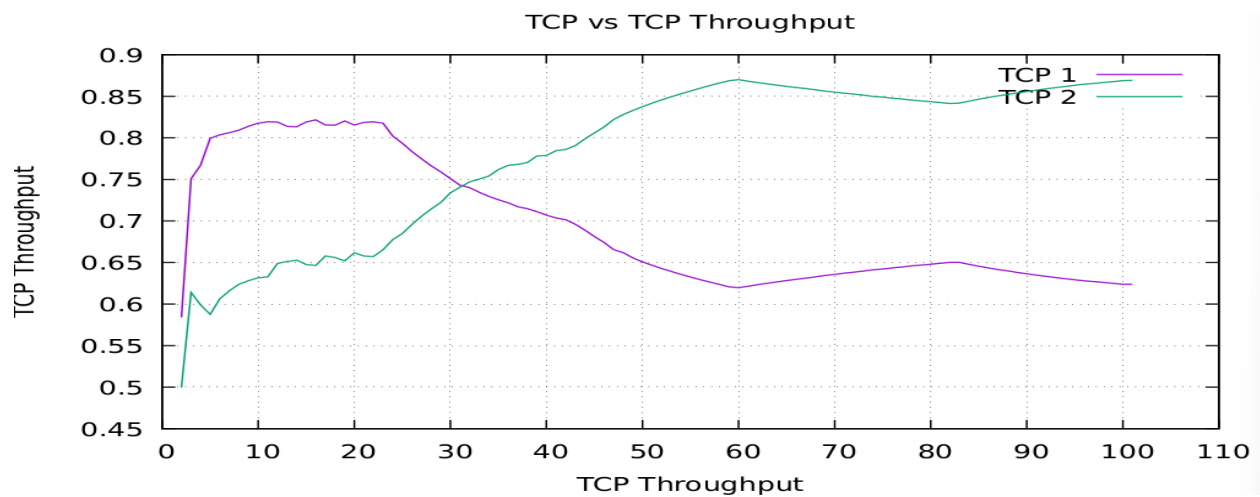


Figure 28: Reno vs Westwood with Data rate of 1 Mbps

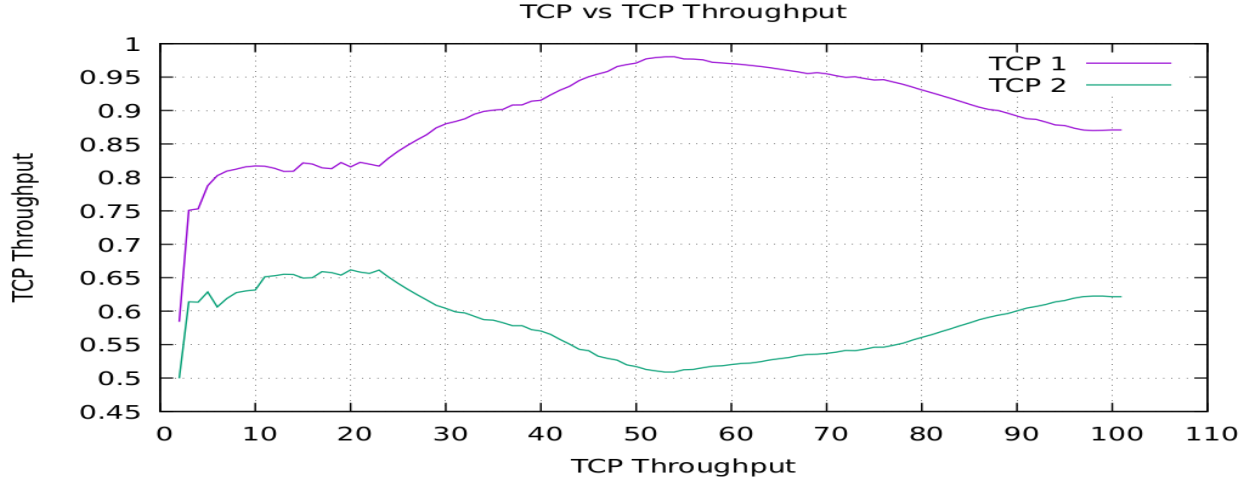


Figure 29: Reno vs Westwood with Data rate of 1.5 Mbps

IV. Conclusion

In this project, we have studied the implementation of the TCP protocol using ns3 simulation. We have explored various flavors of TCP and observed how TCP behavior changes with background traffic. The experiments were conducted on a network with 6 nodes and 5 links, where the link 3-4 was the bottleneck link. The simulations were carried out with default parameter values, and necessary modifications were made according to the problem specification.

In Part 1, we created a single TCP connection between node-1 and node-5 and conducted experiments to observe the operations of TCP. We analyzed the simulation results and studied the impact of modifying queue size on the average throughput. We also observed the impact of congestion control due to packet dropping and timeout events. We plotted congestion window, RTT, EstimatedRTT, and TimeoutInterval graphs to better demonstrate our observations. Furthermore, we investigated the relationship between TCP throughput and packet loss rate by using the rate error model. We plotted the TCP throughput vs. packet loss curve and compared it with the throughput curve from the mathematical analysis. We also studied the effect of flow control by creating a scenario that demonstrated its impact.

In Part 2, we created two transport-layer flows, TCP+UDP and TCP+TCP, and conducted experiments to observe the resource sharing between these flows. We plotted graphs showing the UDP throughput on the x-axis and TCP throughput on the y-axis to determine the fair-share rate of the link. We also calculated the loss rate of the TCP connection by fixing the UDP rate at the fair-share rate. We replaced the drop-tail queue with another queuing scheme provided by ns3 and observed the impact on throughput. We also plotted the throughput of both flows against time and observed if they achieved a fair share of the link. Finally, we studied the effect of different TCP

flavors on resource sharing and plotted graphs to demonstrate our observations. Overall, the experiments conducted in this project provided valuable insights into the implementation and behavior of TCP protocols.

V. Appendix

I have provided a comprehensive collection of all the code files utilized in this project. These files have been organized and consolidated into a single ZIP folder for your convenience. The folder contains all necessary source code and scripts that were developed and employed throughout the project's various stages. This ensures that you have easy access to all the relevant resources for a thorough understanding of the project's implementation and structure.

By extracting the contents of the ZIP folder, you can review, modify, or even execute the code to gain deeper insights into the project's functionality and the underlying mechanisms. The codes are required to be placed in the following folder:

```
cd /ns - allnone - 3.34/ns - 3.34/scratch
```

After placing the scripts, navigate to the same folder in the terminal and run the following command:

```
./waf --run scratch/{file_name}.cc  
gnuplot {filename}_plot.plt
```

VI. Reference

- “Introduction to Doxygen.” ns. Accessed March 25, 2023. <https://www.nsnam.org/doxygen/>.
- “3: Examples/TCP/TCP-Linux-RENO.CC File Reference.” ns. Accessed March 25, 2023. https://www.nsnam.org/docs/doxygen/d1/d7b/examples_2tcp_2tcp-linux-reno_8cc.html.
- “3: Examples/Traffic-Control/Traffic-Control.cc Source File.” ns. Accessed March 25, 2023. https://www.nsnam.org/docs/doxygen/d9/de2/traffic-control_8cc_source.html.
- “3: Src/Internet/Test/RTT-Test.cc Source File.” ns. Accessed March 25, 2023. https://www.nsnam.org/docs/release/3.35/doxygen/rtt-test_8cc_source.html.
- “3: Examples/TCP/TCP-Variants-COMPARISON.CC File Reference.” ns. Accessed March 25, 2023. https://www.nsnam.org/docs/release/3.26/doxygen/tcp-variants-comparison_8cc.html.
- “Error Model¶.” Error Model - Model Library. Accessed March 25, 2023. <https://www.nsnam.org/docs/models/html/error-model.html>.
- “Introduction to Installation of Network Simulator 3 NS3.” YouTube. YouTube, November 24, 2014. <https://www.youtube.com/watch?v=T8NwCPROYYA&list=PLtRbi3COBc5ku7pviNjbU3Z-2ZedBHCP5>.
- “5. ASCII Tracing in NS3- How to Trace Events in Network Simulator 3 (NS3).” YouTube. YouTube, February 4, 2022. <https://www.youtube.com/watch?v=G3AWBwW-HN8>.