Overview

An Ubuntu virtual machine (VM) was set up on a Mac laptop using UTM app. The VM was configured to use a bridge network, which gave the VM its own IP address on the host Mac's LAN. All programs within this report, such as Wireshark, client and server scripts, and iperf, were executed within the VM. A phone with the `TCP/IP Debugger` app was used as a second, portable device. The app was used to create TCP/UDP client and server connections which would trigger files to be transferred.

A TCP client and server script and a UDP server script were implemented in Python using the `socket` library. All three scripts `TCPserver.py`, `TCPclient.py`, and `UDPserver.py` can be found within the `PA2` directory. All pcaps from each stage of this experiment were also added within the `pcaps` subdirectory.

This report aims to analyze the data from the experiment. The `README.md` file within the `PA2` directory has an overview of the specific commands used for each stage of the experiment.

Task One.

1.

The goal for Section 1.1 is to measure how the throughput of TCP and UDP connections over Wi-Fi varies as distance is placed between the client and server. For this part, the TCP and UDP server script was run on the VM, which would send a file to the client connecting from the phone. Three difference distances were accounted for during the experiment: a *close* distance where the phone was located next to the laptop, a *medium* distance where the phone was located in the same building but with moved to a part with weaker Wi-Fi, and a *far* distance where the phone was taken out of the building. 10 files were transferred consecutively for each distance and the recorded throughput is the average bits/sec of the 10 file transfers.

The data was filtered for TCP packets using the `ftp-data || ftp || tcp` filter and for UDP packets using `udp`. Average throughput was obtained through looking at the statistics and graphs of the filtered packets.

The next few figures are the graphs and data for TCP connections. Figure 1 shows some statistics for the packet capture. The average throughput for each of the following graphs was first obtained by looking at the bits/s statistic. Figure 2 shows the instantaneous throughput as files were transferred to the client. The green line shows the average throughput for the duration of the connection.

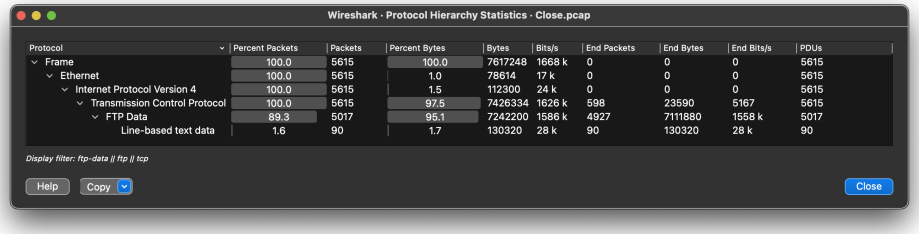

Figure 1: TCP close distance capture - Statistics > Protocol Hierarchy

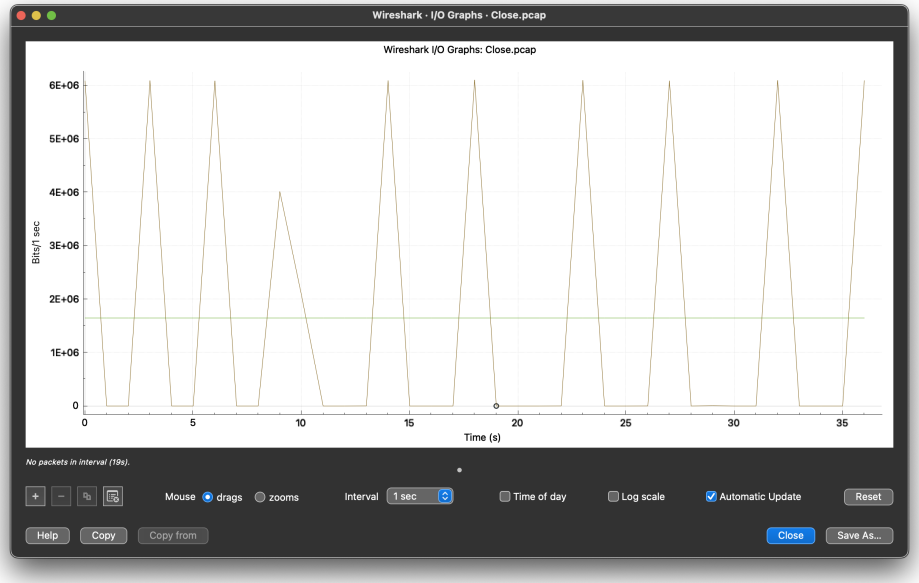

Figure 2: TCP close distance capture - Statistics > I/O Graphs

The figures show that the average throughput of the connection was around 1670k bits/sec. The graph also shows that the instantaneous throughput peaks at around 6000k bits/sec per transfer.

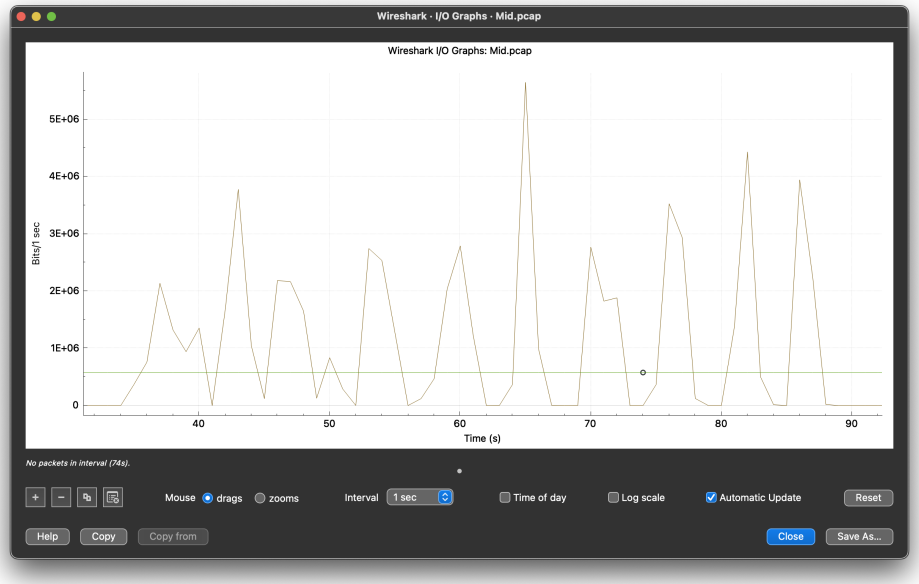

Figure 3: TCP medium distance capture - Statistics > I/O Graphs

Figure 3 shows the instantaneous throughput of a medium distance capture. Its x-axis is scaled to the period where the files were transferred, which could have an effect on the calculated average throughput. It has an average of 580k bits/sec, which is lower than that of the close connection. Notice that the peak instantaneous value of 5500k bits/sec occurs only once throughout the duration of the connection. The peak instantaneous throughput values trend lower at around 3000k bits/sec.

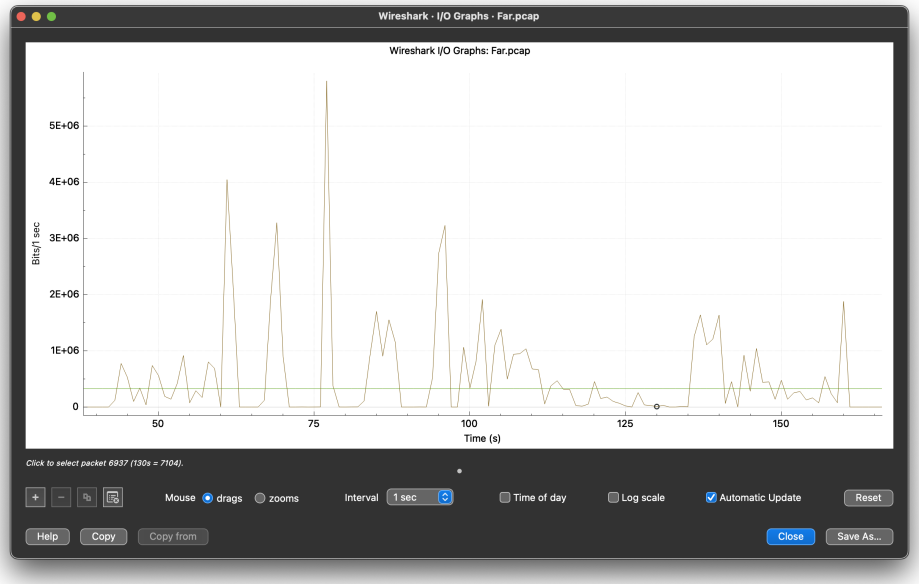

Figure 4: TCP far distance capture - Statistics > I/O Graphs

Figure 4 shows the instantaneous throughput of a far distance capture. Recall that for the far distance capture, the phone was removed from the building and placed outside where the Wi-Fi connection was weakest. This caused a noticable difference in the wait time as each file was transferred. Whereas wait time was negligible the close and medium distances, a longer time was spent between each file transfer for the far distance capture because each file took longer to send.

Comparing the far distance capture to the previous medium and close distance captures, multiple differences can be seen. First, notice that the average throughput of the far distance capture is around 330k bits/sec. This is around one-fifth that of the close distance capture. Also notice that there are less peaks that reach the instantaneous throughput value of 5500k bits/sec. On average, the instantaneous throughput values peak at 2500k bits/sec.

Additionally, when compared to the other two graphs, there are fewer defined peaks for each file transfer. When looking at Figure 2, it is obvious when a file is being transferred because there are 10 defined sharp peaks. This is less so the case for the medium distance capture, but there are still 10 peaks more or less. For the far distance capture, it is difficult to tell when one file transfer ends and when the next one begins.

Distance	**Average throughput (bits/sec)**	**Instantaneous throughput (bits/sec)**
Close	1670k	6000k
Medium	580k	3000k
Far	330k	2500k

For TCP connections, the data shows that a weaker Wi-Fi signal leads to a slower file transfer. The average throughput decreases as the distance between client and server increases.

The next few graphs will be for UDP connections. For UDP connections for this section specifically, files were transferred every 5 seconds utilizing the `TCP/IP Debugger` app and a scheduled delivery feature.

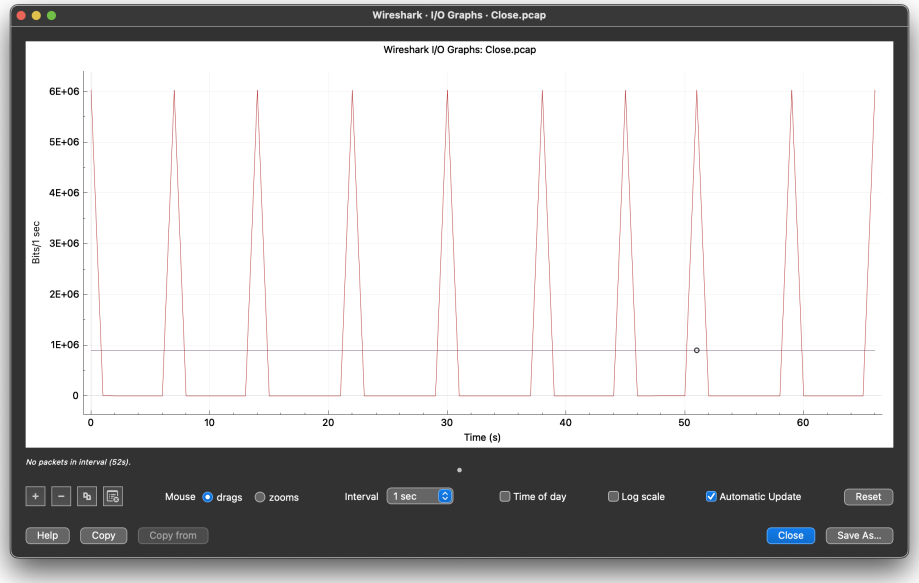

Figure 5: UDP close distance capture - Statistics > I/O Graphs

Figure 5 shows the close distance capture for a UDP connection. The average throughput of the capture is 908k bits/sec, which is lower than the average TCP close distance throughput. However, the peaks for the UDP capture are consistently around 6000k bits/sec. The difference in average throughput could be due to a difference in the overall time it took to send all 10 files, since it took longer for the UDP connection overall.

For the medium and far distance captures, it seems that the distance placed between client and server did not affect the peak throughput of the file transfer. The average throughput for medium and far distances were 430k and 300k bits/sec respectively, but those differences can be attributed to the time it took to physically distance the client from the server.

Note that I accidentally sent 14 files instead of 10 for the far capture and that 2 files sent at the same time for the medium capture. Besides those irregularities, the instantaneous throughput still reaches around 6000k bits/sec. This implies that physical distance does not affect UDP throughput nor cause packet loss.

Distance	**Average throughput (bits/sec)**	**Instantaneous throughput (bits/sec)**
Close	908k	6000k
Medium	430k	6000k
Far	300k	6000k

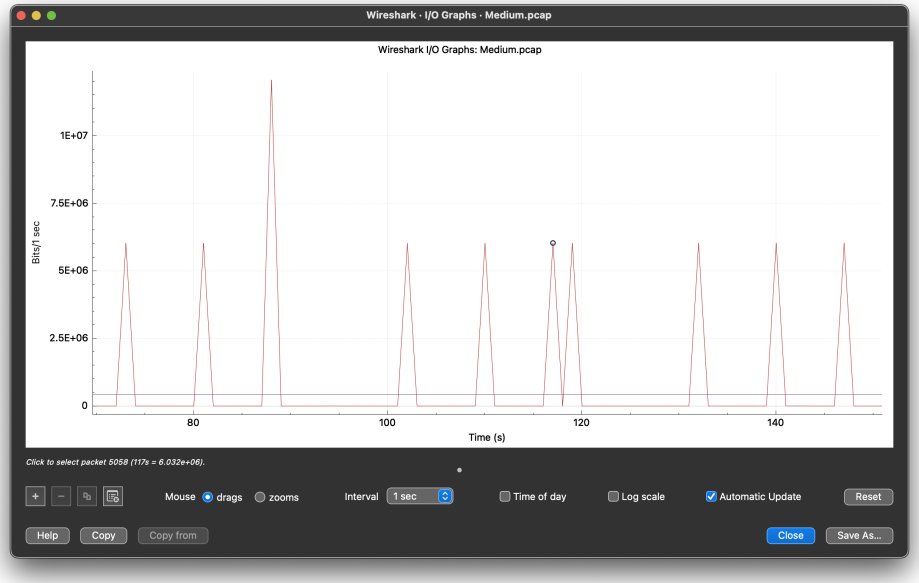

Figure 6: UDP medium distance capture - Statistics > I/O Graphs

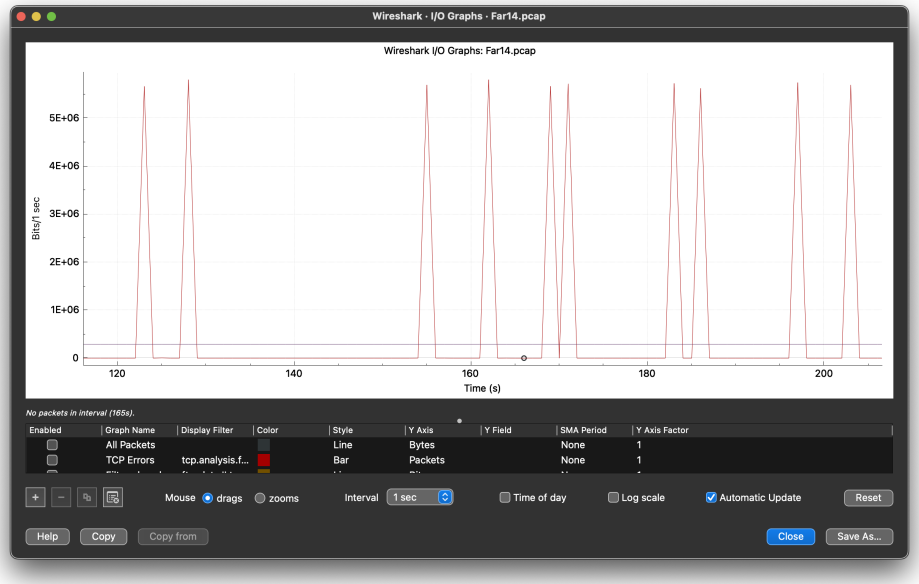

Figure 7: UDP far distance capture - Statistics > I/O Graphs

2.

For Section 1.2, the goal is to repeat the experiment conducted in Section 1.1, but manually controlling the network settings instead of varying the distance between client and server. The laptop acted as the server while the phone acted as the client. Within the VM, `linux-tc` was used to vary network settings and introduce a percentage of packet loss to the network using the command `sudo tc qdisc add dev [enp0s7] root netem loss [\%]`. Three different packet loss values, 1%, 10%, and 30% were tested for TCP and UDP connections. Again, 10 files were transferred consecutively for each packet loss percentage and network traffic was captured using Wireshark.

For TCP connections, data was filtered using the `ftp-data || ftp || tcp` filter. For UDP connections, the filter used was `udp`. Average and instantaneous throughput values were obtained by looking at statistics and graphs.

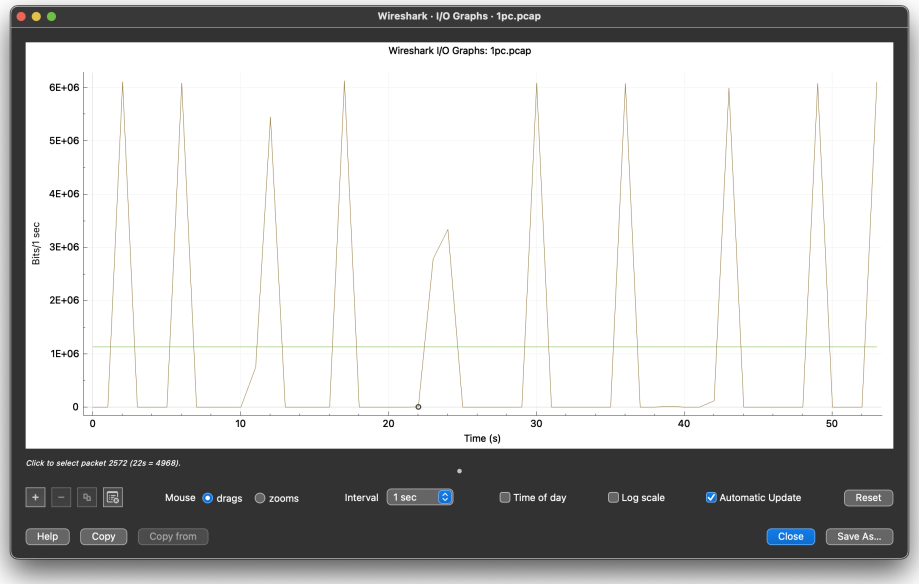

Figure 8: TCP 1% Loss - Statistics > I/O Graphs

Figure 8 has a mandatory 1% packet loss for all traffic leaving the VM. The 1% value was used as a control value, but the average throughput was noticeably only 1200k bits/sec instead of 1600k from the close distance capture. This could be due to the one smaller peak in this packet capture bringing down the average, or the duration of packet capture was longer than that of Figure 2's. Other errors can be accounted for at the end of the report. The instantaneous throughput usually peaked at 6000k bits/sec.

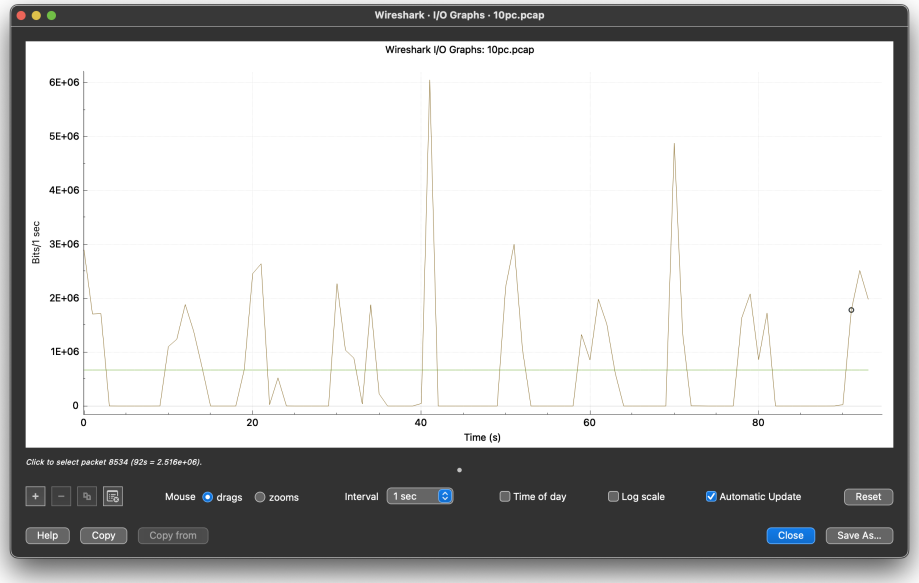

Figure 9: TCP 10% Loss - Statistics > I/O Graphs

Figure 9 shows a 10% packet loss. The average throughput 670k bits/sec, which is comparable the medium distance capture's throughput of 580k bits/sec. It can be roughly estimated that the distance placed between TCP client and server for the medium capture caused around a 10% packet loss. The instantaneous throughput peaks were lower than that of the 1% packet loss, at around 2500k bits/sec, which is again similar to the medium distance capture's data.

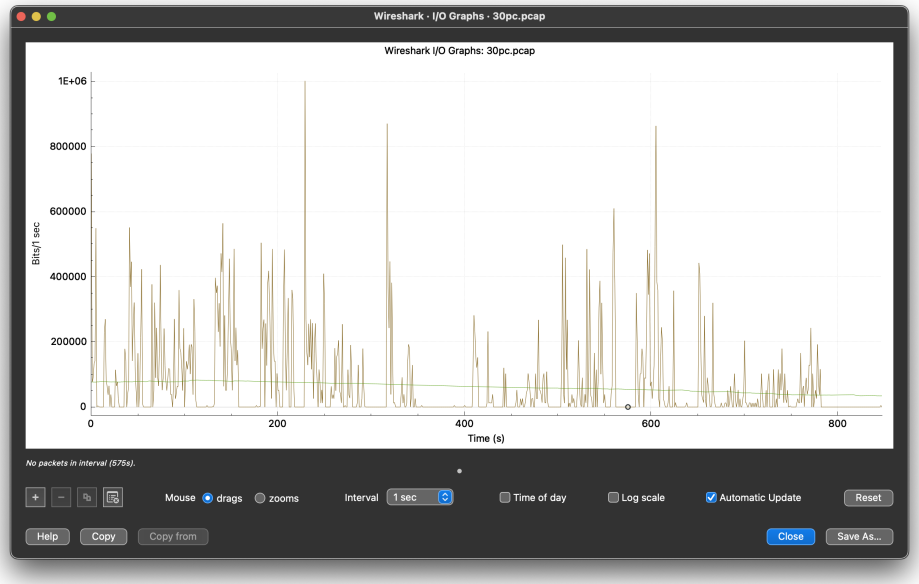

Figure 10: TCP 30% Loss - Statistics > I/O Graphs

Figure 10 shows a 30% packet loss. A 30% packet loss caused a large wait time as each file was transferred, enough such that I could get up from my chair and reorganize my desk and room in the time that it took to send all 10 files. This is notable because all previous captures had either negligible wait times between sending each file, or a few seconds of wait time. A 30% packet loss over TCP for 10 file transfers totaled to multiple minutes.

Like the TCP far distance capture, the 30% loss capture had no noticeable peaks denoting a file transfer. Both the 1% and 10% TCP transfers had noticeable peaks, but the 30% transfer looks like a constant stream of data. The average throughput was only 60k bits/sec, while the highest instantaneous rate did not go over 1000k bits/sec. These values are lower than the 10% and 1% captures, but is expected due to the amount of time spent capturing packets and waiting between file transfers.

% Loss	Average throughput (bits/sec)	Instantaneous throughput (bits/sec)
1	1200k	6000k
10	670k	2500k
30	60k	1000k

In general, the TCP data shows a correlation between distance from client to server and dropped packets, both of which lead to lower average throughputs. It can be assumed that longer distances between TCP client and server imply a weaker Wi-Fi signal, which causes packet loss, which leads to a decreased throughput.

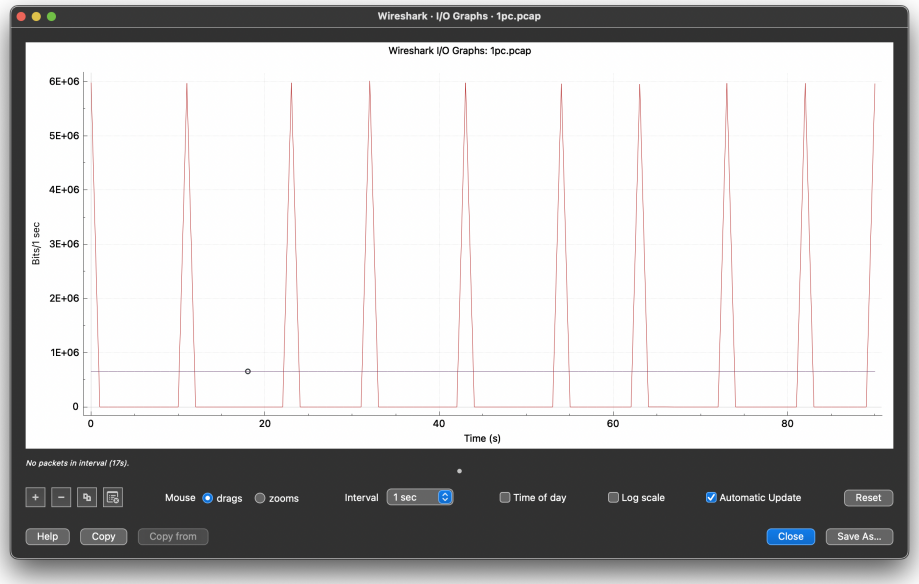

Figure 11: UDP 1% Loss - Statistics > I/O Graphs

Figure 11 shows a 1% packet loss for a UDP connection. The average throughput for the capture was 660k bits/sec, which is lower than the previous TCP 1% capture and also lower than the UDP close distance capture. This is due to an inconsistency in time between when the file transfer finishes and the next one begins. Notice that the instantaneous throughput peaks at 6000k bits/sec, which is the same value of the peaks of the TCP 1% capture and the UDP close distance capture from Section 1.1.

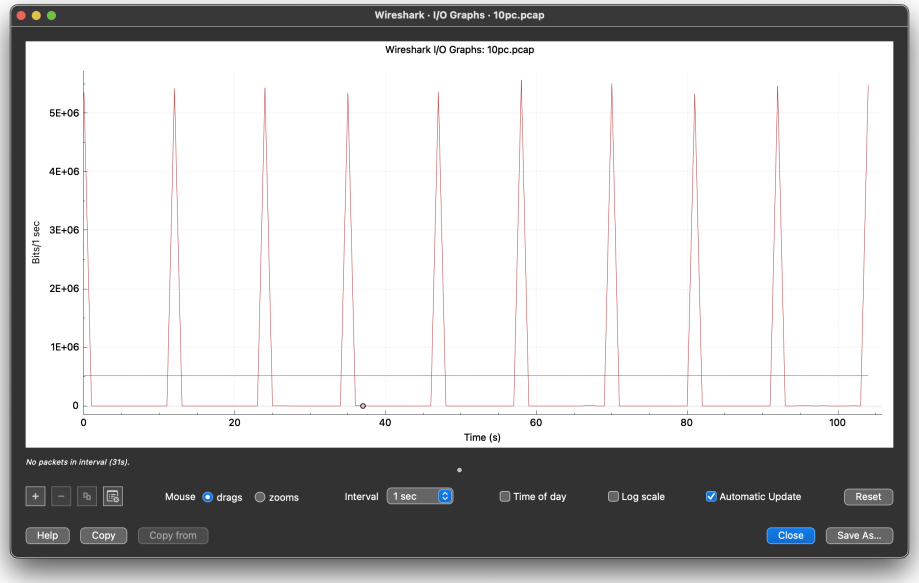

Figure 12: UDP 10% Loss - Statistics > I/O Graphs

Figure 12 shows a 10% loss for a UDP connection. The average throughput was 520k bits/sec and the instantaneous peaks were around 5300k bits/sec. Comparing the UDP 10% loss to the UDP 1% loss graph, there is no change the shape of the peaks of each file transfer. Recall for TCP connections, as throughput decreased, it became harder to tell when one file transfer ended and another one started. For UDP connections, the packet loss seemed to have no effect on the rate at which packets were transferred. However, the instantaneous rate peaked lower, which could have been influenced by packet loss since there were less packets to transfer.

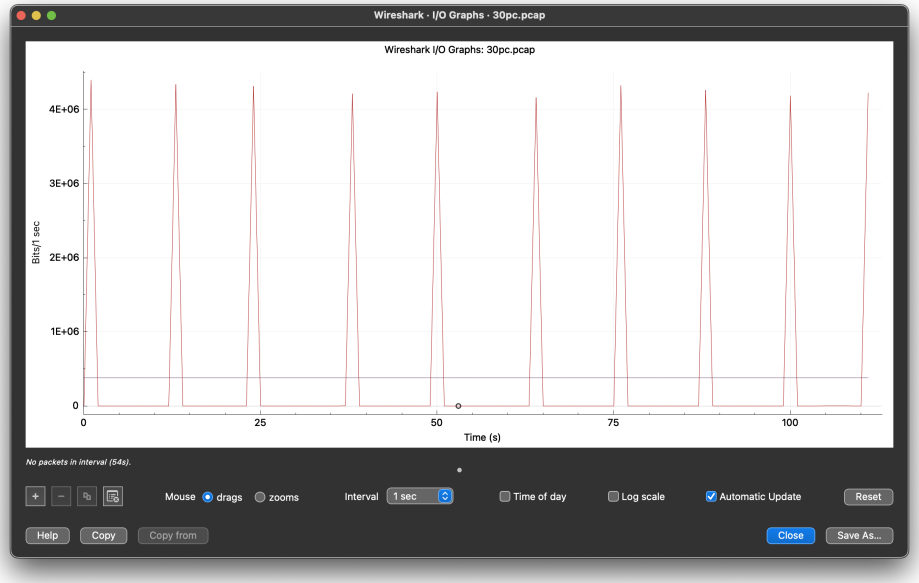

Figure 13: UDP 30% Loss - Statistics > I/O Graphs

Figure 13 shows a 30% loss for a UDP connection. The average throughput was 385k bits/sec and the instantaneous peaks were around 4250k bits/sec. Notice that there is no deformity in the shapes of the peaks and that each file transfer start and end is clearly defined. The packet loss introduced to the network had no effect on the speed of the packet transfers, which is different than the results from the TCP connections.

Recall that packets sent using UDP transport protocol are not guaranteed to arrive at their location. UDP protocol prioritizes speed over guaranteed packet arrival, while TCP prioritizes packet arrival. This is reflected in the 30% loss graphs for TCP and UDP. The average throughput for TCP was lower because there were many dropped packets that had to be retransmitted, while UDP sent off the packets with no regard for which ones arrived, leading to a higher throughput compared to TCP at the same loss percentage.

% Loss	TCP Average	TCP Instantaneous	UDP Average	UDP Instantaneous
1	1200k	6000k	660k	6000k
10	670k	2500k	520k	5300k
30	60k	1000k	385k	4250k

3.

The goal in Section 1.3 is to vary the network settings and analyze how it changes network performance, and select the best settings for the network. Four parameters were tested for TCP connections: receive window size, send window size, TCP flavor, and window scaling. For UDP connections, two parameters were tested: max send window size and max receive window size. For each parameter, two different packet captures were taken, each with 10 file transfers with 5 second wait times between the end of a file transfer and the start of the next transfer. For all parameters and the control captures, a mandatory 5% packet loss was introduced to the network to allow for more data to be collected on TCP window size, although it is unsure if this was effective. For all captures, the VM acted as the server while the phone acted as a client.

Instead of importing graphs like the previous sections, all data values will be recorded into a table to compare alongside each other. This is because there are too many packet captures and graphs in this section to include in the report. All packet captures are available in the PA2 directory for more analysis.

For TCP connections, each parameter was changed in the following way before being reset to the original value:

1. Window scaling: default value = 1 $\rightarrow$ 10.
2. TCP flavor: default = cubic $\rightarrow$ reno
3. Receive window: default = $[4096, 131072, 6291456] \rightarrow [131072, 131072, 6291456]$
4. Send window: default = $[4096, 16384, 4194304] \rightarrow [16384, 16384, 4194304]$

	Control	**Window scaling**	**TCP flavor**	**Receive window**	**Send window**
Average throughput	946k / 988k	1090k / 1050k	1080k / 1077k	1005k / 946k	1046k / 1001k
Instantaneous throughput	4000k / 4500k	4250k / 3500k	4500k / 4500k	5000k / 4000k	4500k / 4000k
Streams	10(10) / 10(11)	10 / 10	12 / 10	10 / 11(16)	11(15) / 10(12)

The table above shows the data for all 10 trials of the TCP packet captures. In general, the instantaneous throughput values vary amongst each parameter. Not much can be derived from these values. The average throughput values vary more, but not by much. It seems that all of the parameters had an effect on the network, but increasing the window scaling factor and changing the TCP flavor to Reno had the largest effect. The effect of using TCP Reno will be explained in the next section.

The average throughputs of receive and send window sizes are about the same. Each parameter was tested in isolation, which meant that at no point were the send and receive windows both different from their default values. This could have affected their throughput since an increased send window would work well with an increased receive window, but the default size would have caused a bottleneck and allowed for more dropped packets. Interestingly, increasing one parameter still led to an increase in average throughput from the control. I am unsure why this is the case. Window scaling also seemed to have a decent effect on throughput, since increasing window scale factor meant that it would be easier to recover from packet losses.

For UDP connections, the send window size parameters were grouped together and tested, and the receive window parameters were grouped together and tested:

1. Receive window (default): default = 212992 → 26214400
2. Receive window (max): default = 212992 → 26214400
3. Send window (default): default = 212992 → 26214400
4. Send window (max): default = 212992 → 26214400

	Control	**Receive window**	**Send window**
Average throughput	1014k / 997k	972k / 920k	639k / 1007k
Instantaneous throughput	5750k / 5750k	5700k / 5750k	5725k / 5750k

Notice that the instantaneous throughput for all three parameters are the same. The average throughput varies a little bit and it seems that changing the receive and send windows led to lower average throughput. So far, it seems changing any parameters such as distance or send and receive window sizes has not let to any noticeable increase or decrease in performance for UDP connections. The only change that has had an effect is the percentages of packet loss introduced to the network in Section 1.2.

4.

In Section 1.4, the goal is to plot the throughput and congestion values from Section 1.3 and guess the TCP flavor. Unfortunately, TCP flavor was one of the varied parameters tested within Section 1.3, which meant that I already knew the TCP flavor was Cubic. However, the control packet captures and the Reno packet captures could be analyzed to look for differences in throughput.

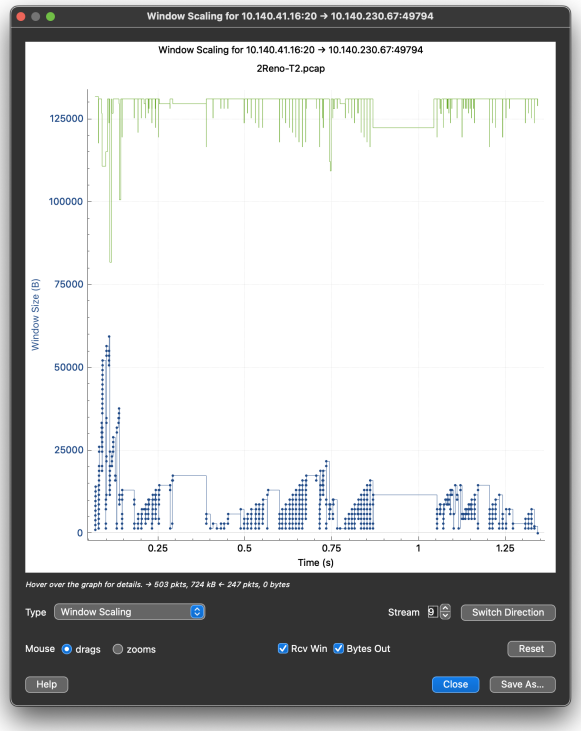

Figure 14: Reno - Statistics > TCP Stream Graphs > Window Scaling

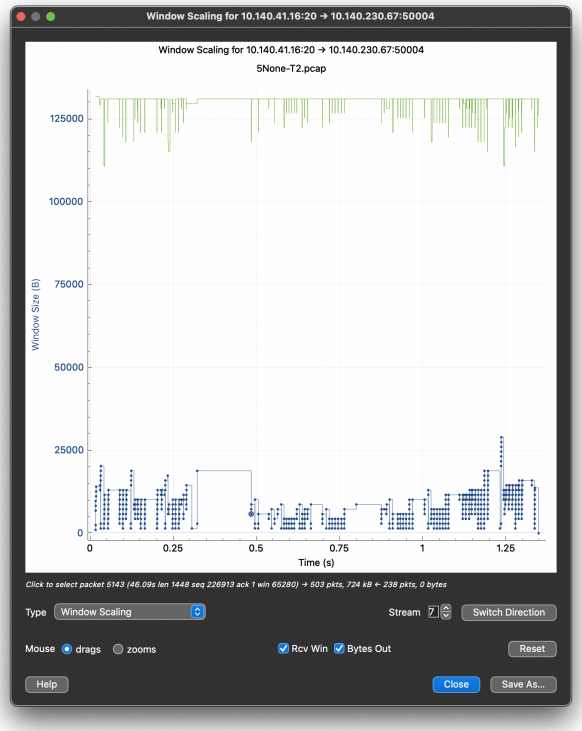

Figure 15: Cubic - Statistics > TCP Stream Graphs > Window Scaling

Notice that the Reno graph has a linear window size increase. The Cubic graph does not seem to have much increase at all. This is due to the 5% packet loss introduced earlier in Section 1.3. TCP Cubic will increase the window size according to that of a cubic function, but it takes time for the window size to reach a point where the increase can be observed on a graph. The 5% packet loss forced the window size to reset often, which is why there is minimal visible growth in the Cubic graph. This is also the reason why the Reno captures had higher average throughput than the control Cubic captures. With a 5% loss introduced to the network, TCP Cubic has no time to take up resources according to its cubic increase.

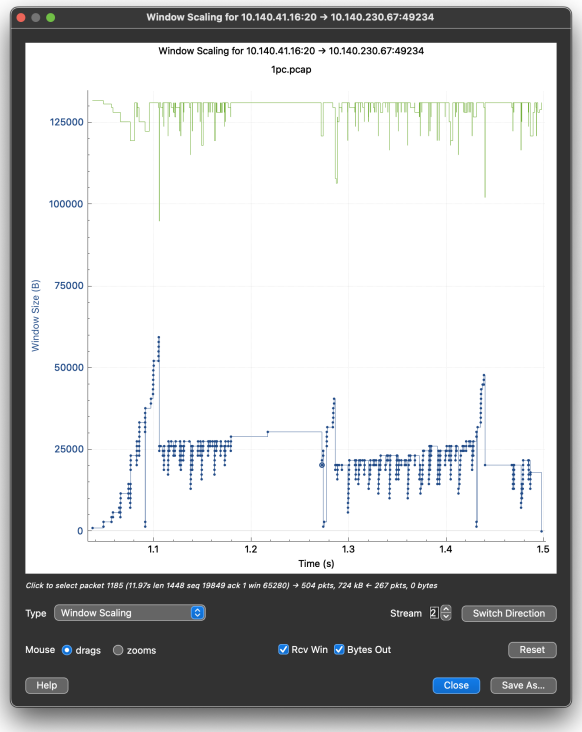

Figure 16: TCP 1% Loss - Statistics > TCP Stream Graphs > Window Scaling

Above is the window scaling for a 1% loss TCP packet capture from Section 1.2. Notice the increase of the window size follows the latter half of a cubic graph more closely than the previous cubic control capture did. It also shows the slow phase of the cubic function after recovering from a packet loss.

To conclude this section, the TCP flavor used within the VM was TCP Cubic. This was concluded while testing parameters within Section 1.3, but there are other details that arise from comparing TCP stream graphs, that point towards TCP Cubic being the correct algorithm.

Part Two.

The goal for Section 2 is to measure the bandwidth of the network using `iperf`. An `iperf` server was established using iOS app `WifiPerf Endpoint` to which the VM connected to using the command `iperf -c 10.140.230.67 -t 30 -i 1`. The test ran for 30 seconds and took a snapshot of the bandwidth every 1 second. This test was run a total of 10 times.

	Bandwidth (Mbps)
1	90.7
2	91.1
3	93.3
4	66.5
5	92.8
6	78.8
7	90.1
8	92.8
9	92.7
10	94.1

The average bandwidth was 88.38 Mbps, which is equal to 707000k bits/sec. The highest instantaneous throughput achieved in Task One was around 6000k bits/sec, or around 0.8% of the bandwidth. It can be estimated that the maximum achievable throughput is equal to $\frac{\text{TCP window size}}{\text{RTT}}$, which means that the limiting factor was most likely the TCP window size. In Section 1.3, the TCP send and receive window sizes were changed, which led to a small increase in performance, but not by more than a few tenths of a percent.

In previous sections, the TCP and UDP throughputs were measured. In Part Two, `iperf` measured the bandwidth of the network, which is different than throughput. Bandwidth is the maximum amount of data that could travel from one device to another over the network. But realistically, the throughput cannot be that high of a value. There are many other factors that effect the TCP throughput, such as the size of the file being transferred, how many users are currently transferring files over the network, distance between client and server, or packet loss within the network. These factors prevent the throughput from taking over the network's resources, which is why the measured bandwidth is many times larger than the experimental throughput values.

Errors.

During each stage of the experiment, there was a lot of room for errors to occur while I was capturing packets, establishing connections, or parsing data. This section aims to address some of the errors that I believe affected the data and how I would correct for those errors if repeating this experiment.

Primarily, there was a large difference in the total runtime of each packet capture. All of the captures were taken over the course of a week, where I gradually grew aware that the average throughput calculated in Wireshark is affected by the total runtime of the packet capture. I tried to correct for this by requiring a 5 second buffer time between the end of a file transfer and the beginning of the next one, but I believe this only added more error to the data. The error is most present in Section 1.1 where time was taken to physically walk a distance away from the laptop. The time taken to move to and from the laptop decreased the average throughput for medium and far captures by a significant amount, as shown in the UDP captures for that section. To correct for this in future attempts, I think the best way would be to leave no buffer time between the end and start of file transfers as well as between the start of packet capturing and transfer of first packet, and the transfer of the last packet and the end of packet capturing. My reasoning behind this is that the downtime between file transfers skews the average throughput, which can be avoided.

Second was the 5% packet loss I introduced in Section 1.3. Initially my intent was to allow for more data to be captured on how the window size increases, but I believe that the mandatory packet loss caused more issues than it helped. This is evident as I explained in Section 1.4 where it appeared that TCP Reno was more effective than TCP Cubic since there were so many packet losses. On one hand, I believe changing the packet loss to 1% could make a difference, but I am more inclined to get rid of the mandatory packet loss as a whole and let the network run by itself. Additionally, the data for most of Section 1.3 seemed weak. I'm not sure if this could be fixed with more trials, or maybe more file transfers per packet capture, or by sending larger files, all of which could have an effect on the average throughput.

Another issue I noticed was the behavior of the network during different locations or different times of the day. I did my packet captures at various places on campus, including [redacted] I ran `iperf` tests at these locations at different times of day, and I found that the average bandwidth varied from 50 Mbits/sec to 150 Mbits/sec. This would have had an effect on the data, but I'm not sure how to correct for it. It seems that one option would be to do all packet captures within the same location at the same time each day to minimize differences in network traffic.

Finally, for UDP, I briefly considered that there would be a difference between using the laptop as a client and server. For all UDP connections, the laptop was sending the file to phone, but the laptop was also the server. For TCP connections I don't think this made a difference because TCP is connection-based, but for UDP I think I lost some data because I tracked throughout of the data being sent out and not received. For example, there could have been packet drops after the packets were sent out, but went unreported because the phone acted as the receiving client.