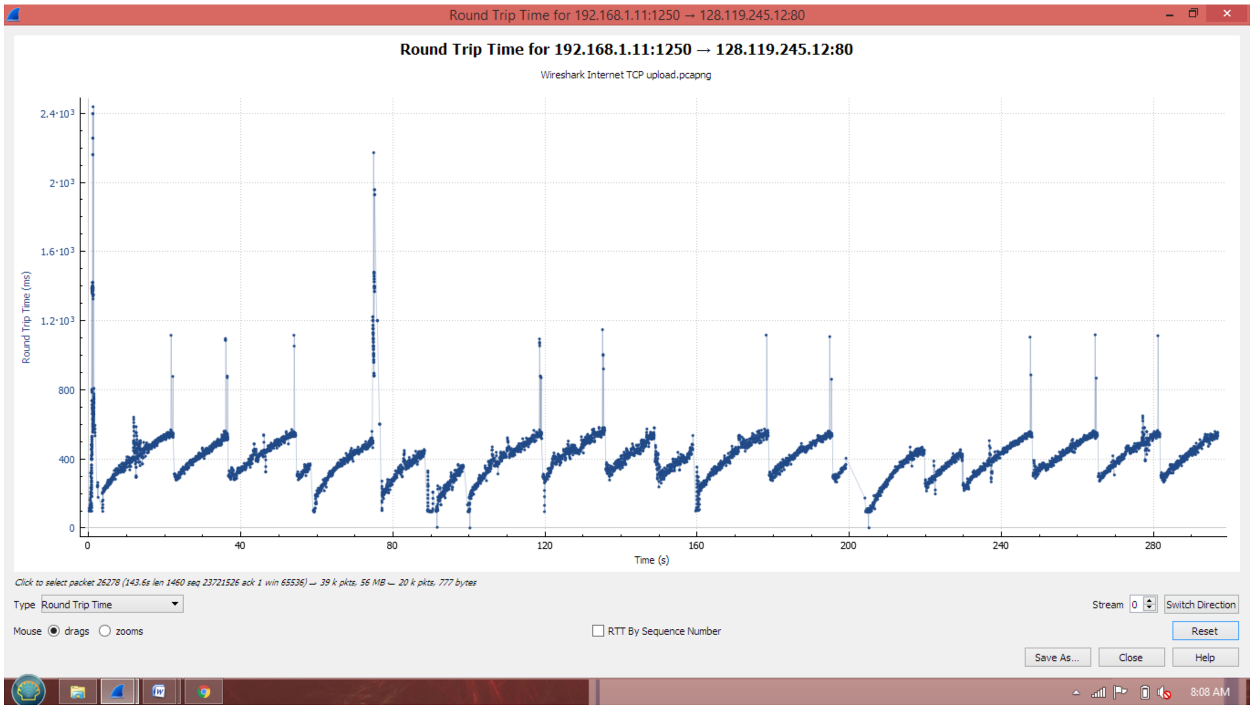
COMP 7005 Assignment 2

Section A

Wireshark Internet TCP Upload (Client-side)

Note: this section was done using an internet connection from home and an upload to the site gaia.cs.umass.edu. Also the graphs being used are from using Wireshark 2.4.2

1) **Plot a TCP Round Trip Time Graph and provide a clear and detailed explanation of what the graph is indicating.**



Click to select packet 26278 (143.6s len 1460 seq 23721526 ack 1 win 65536) → 39 k pkts, 56 MB ← 20 k pkts, 777 bytes

The graph above shows an initial spike which could be from the connection being established. Following this is what looks like data being sent in a gradually increasing amount before the amount which can be sent gets cut down to about half. This could be from the TCP upload being throttled due to the buffer size reaching its limit. The spikes near the top of the gradual increase could be due to the packets being lost which results in the cut in the window size. This could also be due to internet traffic and some point in the connection to the server which could also cause packet loss, resulting in the window size being cut.

2) **Select a set of 6 to 10 segments and analyze the difference between when each TCP segment was sent, and when its acknowledgement was received. Given your analysis, what is the RTT value for each of the segments you have selected? What is the EstimatedRTT value after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT**

for the first segment, and then is computed using the EstimatedRTT equation on page 249 for all subsequent segments.

*EstRTT = (1 – α)*estimatedRTT + α*sampleRTT*

Packet 438) 0.099146000	ACK = p442
Packet 439) 0.110264000	ACK = p445

Est = (1 – 0.125)*0.099146000 + 0.125*0.110264000
Est = 0.08675275 + 0.013783
Est = 0.10053575

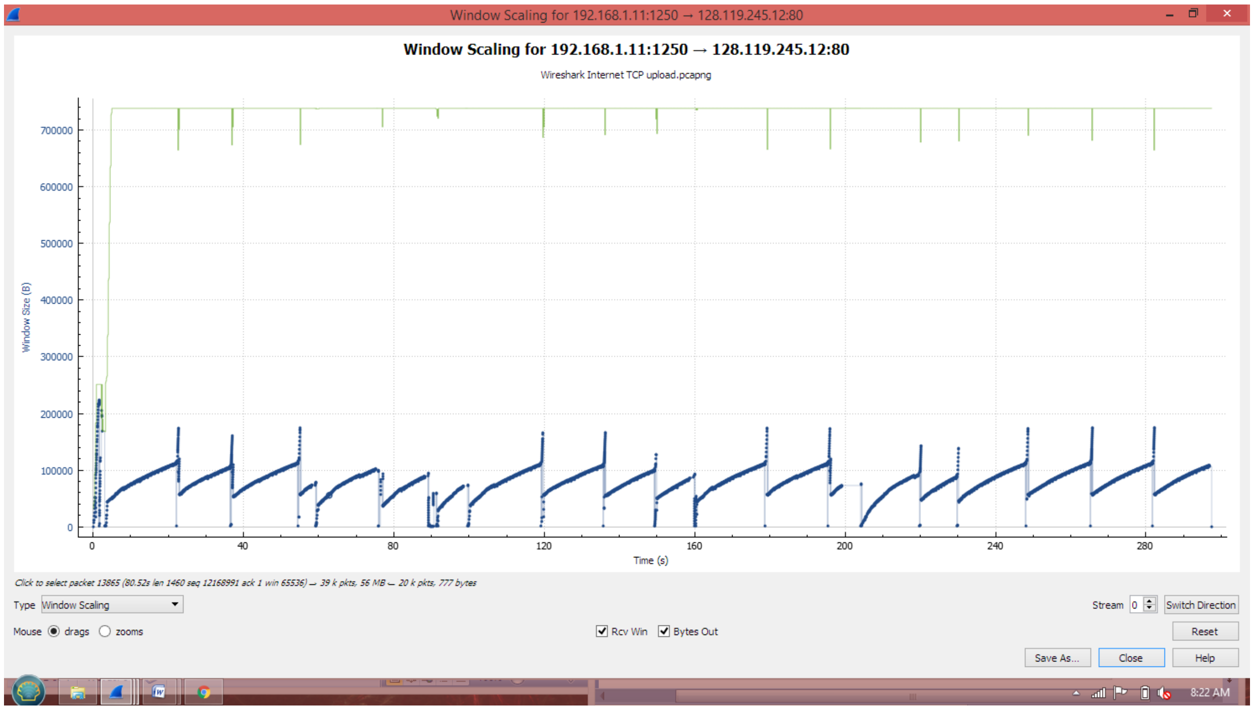
Packet 440) 0.112112000	ACK = p448
Packet 441) 0.114490000	ACK = p451

Est = (1 – 0.125)*0.112112000 + 0.125*0.114490000
Est = 0.098098 + 0.01431125
Est = 0.11240925

Packet 443) 0.099184000	ACK = p455
Packet 444) 0.107595000	ACK = p458

Est = (1 – 0.125)*0.099184000 + 0.125*0.107595000
Est = 0.86786 + 0.013449375
Est = 0.881309375

3) What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender? Plot the Window size versus time for each of the data sets and comment on the results.

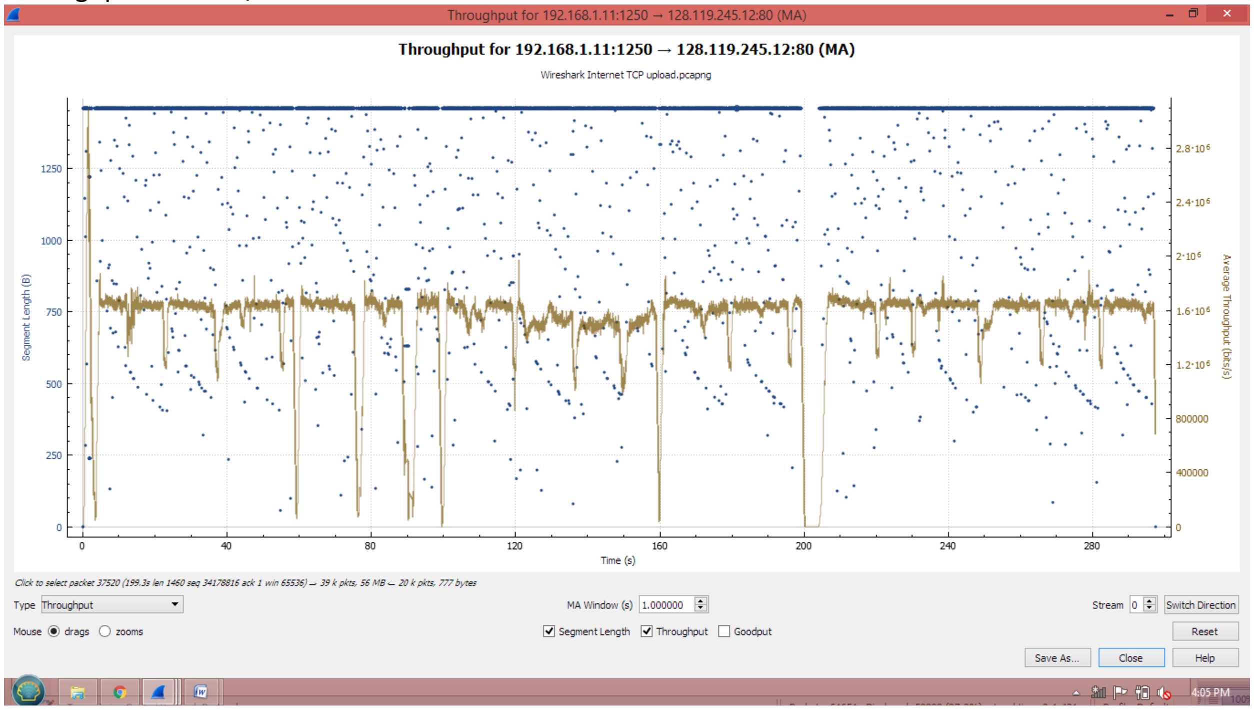


Click to select packet 13865 (80.52s len 1460 seq 12168991 ack 1 win 65536) → 39 k pkts, 56 MB ← 20 k pkts, 777 bytes

The initial SYN, ACK segment has a window size of 29200. A lack of buffer space affects the transmission as small spikes can be seen near the ends of each gradual increase to the window size before a cut down to roughly half of the window size is done. Throttling is evident due to this, leading to the graph having a saw-tooth like effect.

4) What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

The size of the file sent was 55MB, or 55000kb. The total time to transfer the file was approximately 297.5 seconds. Using this data, the throughput was about 184.87394957 kilobytes / second
Throughput = 55000/297.5


Throughput for 192.168.1.11:1250 → 128.119.245.12:80 (MA)

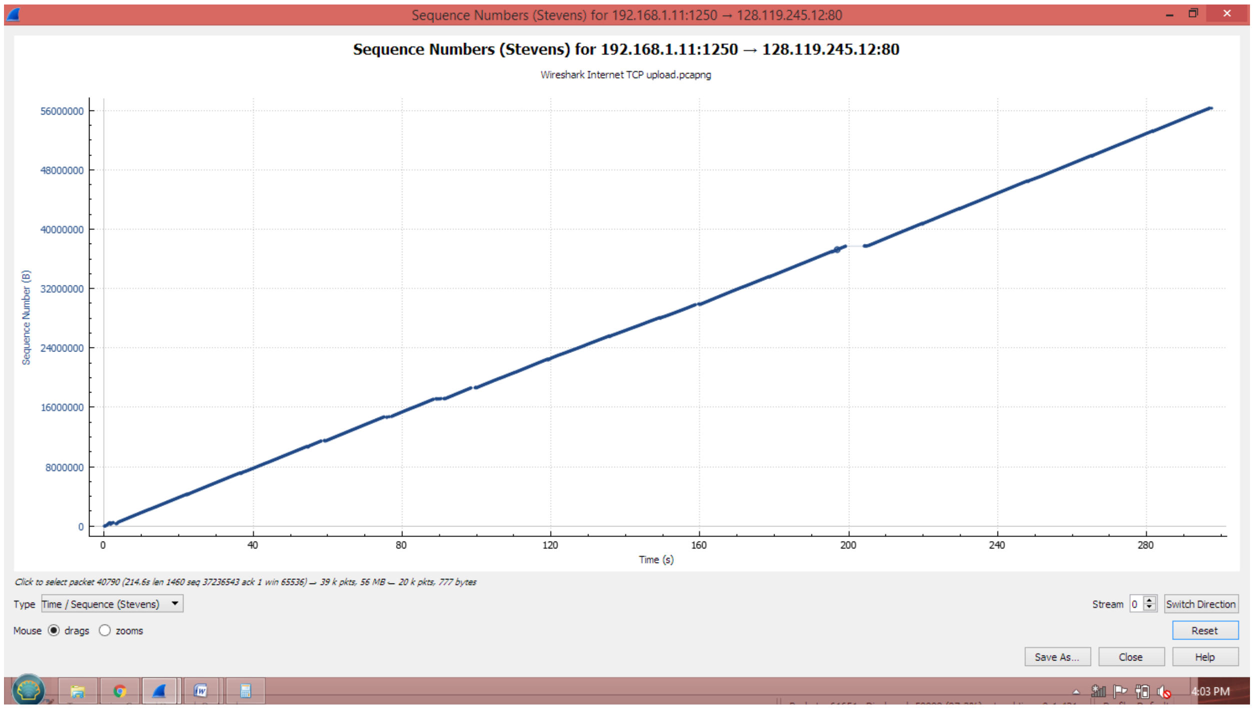
Throughput for 192.168.1.11:1250 → 128.119.245.12:80 (MA)

Wireshark Internet TCP upload.pcapng

Click to select packet 37520 (199.3s len 1460 seq 34178816 ack 1 win 65536) → 39 k pkts, 56 MB ← 20 k pkts, 777 bytes

Type: Throughput
MA Window (s): 1.000000
Stream: 0
Mouse: [✓ drags] zooms
☑ Segment Length ☑ Throughput ☐ Goodput

5.) Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the one machine to another. Identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we have discussed in lectures.



Click to select packet 40790 (214.6s len 1460 seq 37236543 ack 1 win 65536) → 39 k pkts, 56 MB ← 20 k pkts, 777 bytes

Type: Time / Sequence (Stevens)

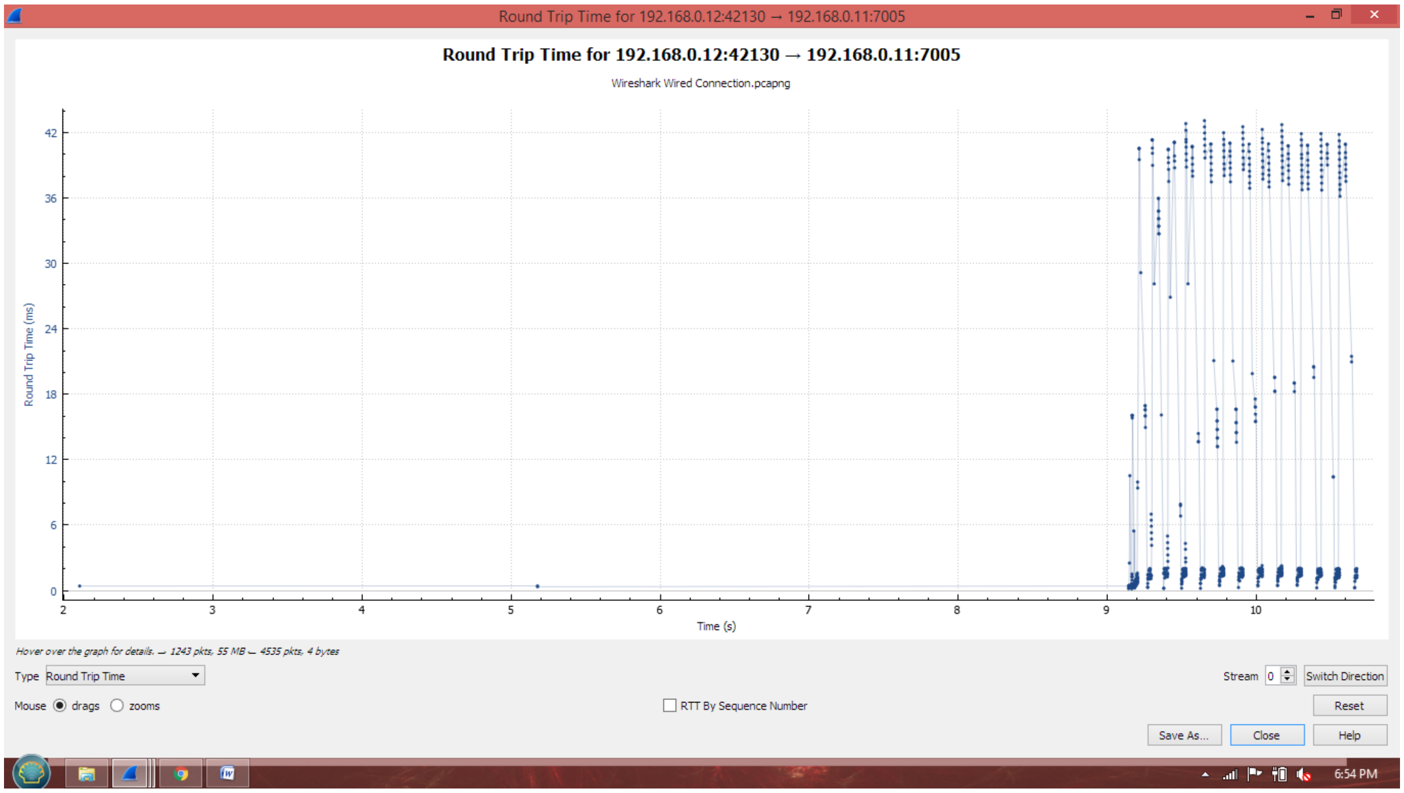
Mouse: (●) drags ( ) zooms

The Slow Start begins right around the start, and quickly hits the point where congestion may have happened that causes a break in the graph. Soon after congestion avoidance takes place, the results are displayed in a fairly linear growth. There are still small breaks, which could be from packets being dropped or larger segments of data being sent. In particular there is a very noticeable breakpoint, where for several seconds the sequence number isn't increasing at all. This could be due to the connection being lost, which would cause the confirmation ACK of sent data to never be returned for the sender to start sending the next set of data. In regards to the idealized behaviour of TCP, this graph is fairly on point with how the optimal TCP behaviour would be with the exception of the large break in the graph.

Wireshark Wired Connection TCP (Client-side)

Note: this section was done using the TCP server + client executables from assignment 1 in Linux on lab machines. Also the graphs being used are from using Wireshark 2.4.2.

1) **Plot a TCP Round Trip Time Graph and provide a clear and detailed explanation of what the graph is indicating.**



The Round Trip Time graph above shows a start from which is the initial connection from the client executable we used, were we would send a connection attempt to the server. There is a pause after the initial dot which would have been entering the command to send a file via TCP. Afterwards the second line dot which would have been when the client had established a connection then send a SEND command to the server with the file name. The next pause between dots would have then been the server received the request to SEND and would either confirm or deny it, in this case confirmed as there is the series of dots afterward.

There is a series of dots that are closely grouped together which could have been the packets being sent as a batch to the server. The small jumps every now could have been from a larger section of data being sent to the server. The large jumps could have been the connection being congested or packets being lost. Going by the larger cluster of dots, the RTT of the wired connection is relatively small and is consistent.

2) **Select a set of 6 to 10 segments and analyze the difference between when each TCP segment was sent, and when its acknowledgement was received. Given your analysis, what is the RTT value for each of the segments you have selected? What is the EstimatedRTT value after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 249 for all subsequent segments.**

*EstRTT = (1 – α)*estimatedRTT + α*sampleRTT*

Packet 50) 0.000407923 ACK = p57
Packet 52) 0.000332562ACK = p58

EstRTT = (1 – 0.125)*0.000407923 + 0.125*0.000332562
EstRTT = 0.000356932 + 0.00004157
EstRTT = 0.000384157

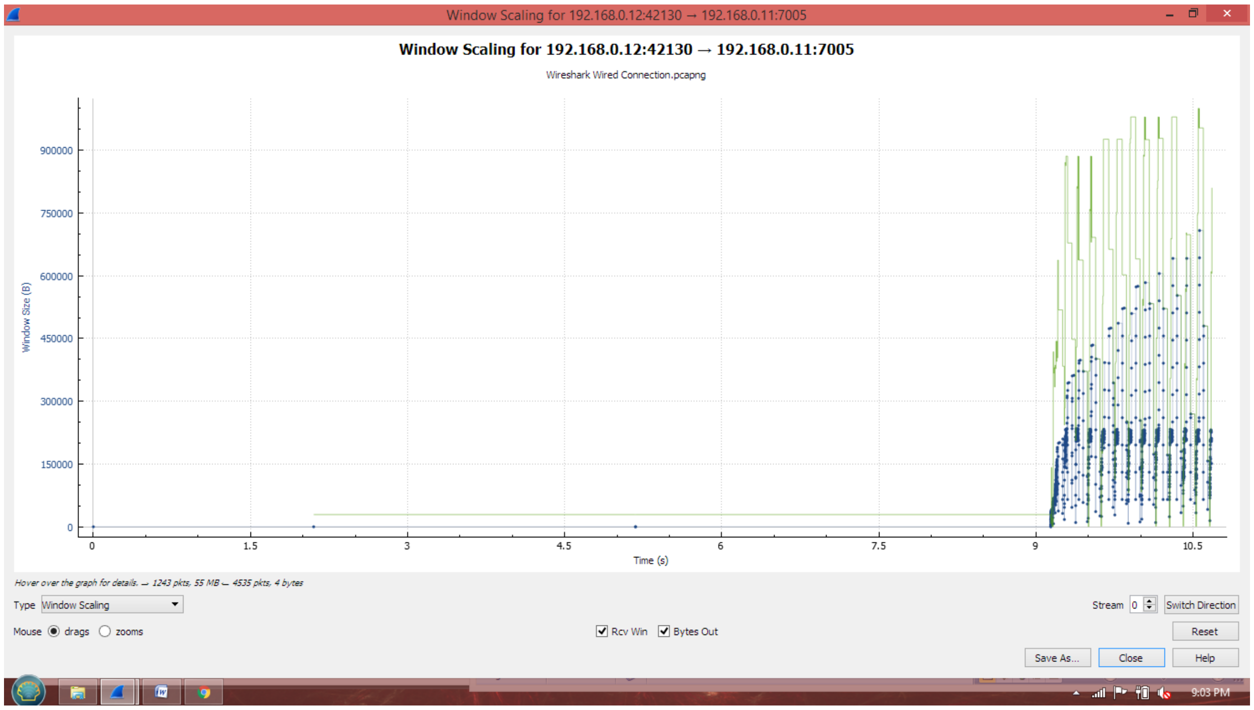
Packet 53) 0.000258682	ACK = p59
Packet 56) 0.000167207	ACK = p61

EstRTT = (1 – 0.125)*0.000258682 + 0.125*0.000167207
EstRTT = 0.000226346 + 0.0000209
EstRTT = 0.000247246

Packet 65) 0.000447240	ACK = p73
Packet 67) 0.000401487	ACK = p74

EstRTT = (1 – 0.125)*0.000447240 + 0.125*0.000401487
EstRTT = 0.000391335 + 0.000050185
EstRTT = 0.00044152

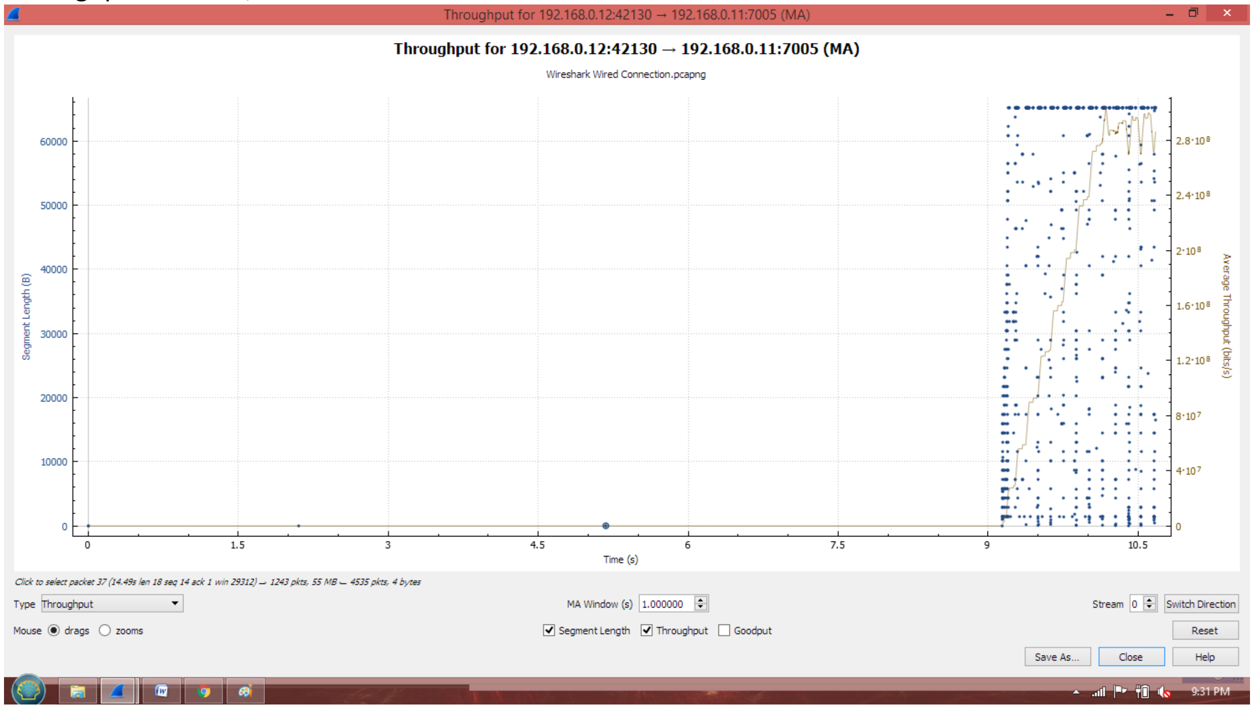
3) What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender? Plot the Window size versus time for each of the data sets and comment on the results



The initial buffer space was 28960, shown under TCP in the initial SYN, ACK exchange. Evidently there was some throttling of data from the sender, as some segments would not much buffer space left before a cut would happen. Each time this happens the upper limit would be slightly higher until the final packet segment is sent.

4) What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

The file size that was send was 55MB, or 55000kb. The total time to transfer the file was approximately 1.5 seconds. Using this data, the throughput was about 36,666.66667kilobytes / second
Throughput = 55000/1.5

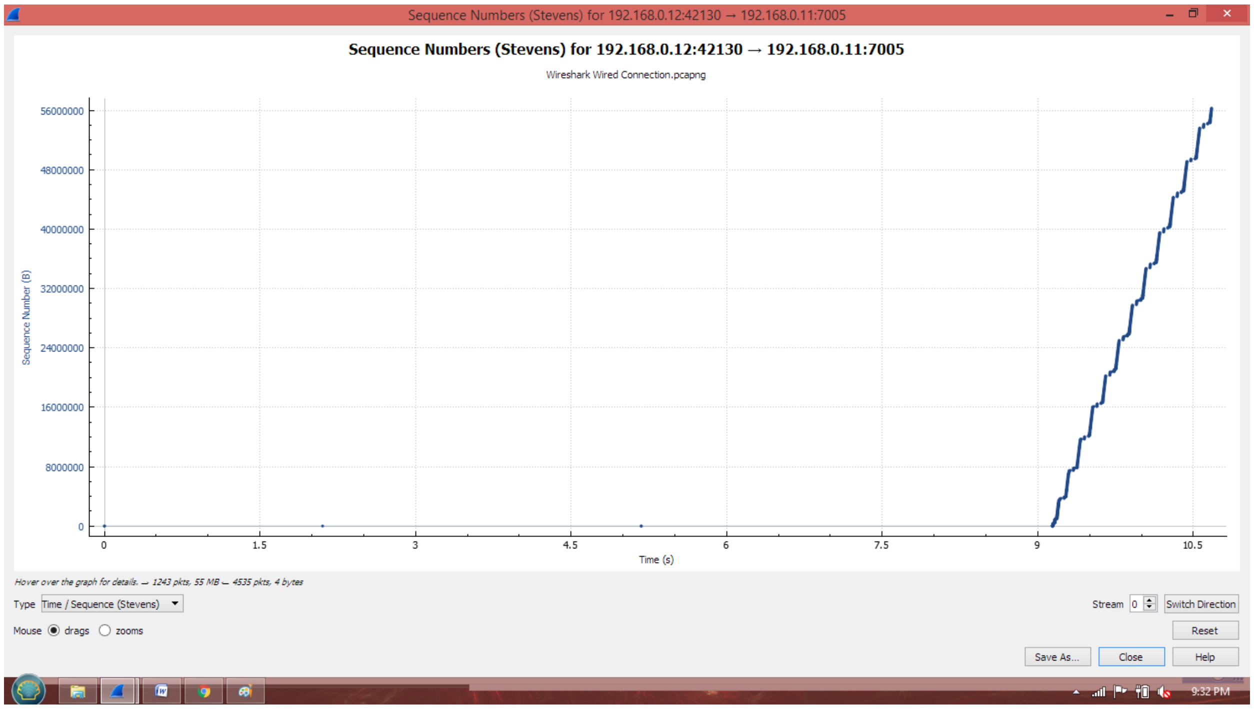

Throughput for 192.168.0.12:42130 → 192.168.0.11:7005 (MA)

Wireshark Wired Connection.pcapng

Click to select packet 37 (14.49s len 18 seq 14 ack 1 win 29312) — 1243 pkts, 55 MB — 4535 pkts, 4 bytes

Type: Throughput
MA Window (s): 1.000000
Stream: 0
Mouse: [✓ drags] zooms
[✓] Segment Length [✓] Throughput [ ] Goodput

5) Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the one machine to another. Identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we have discussed in lectures.

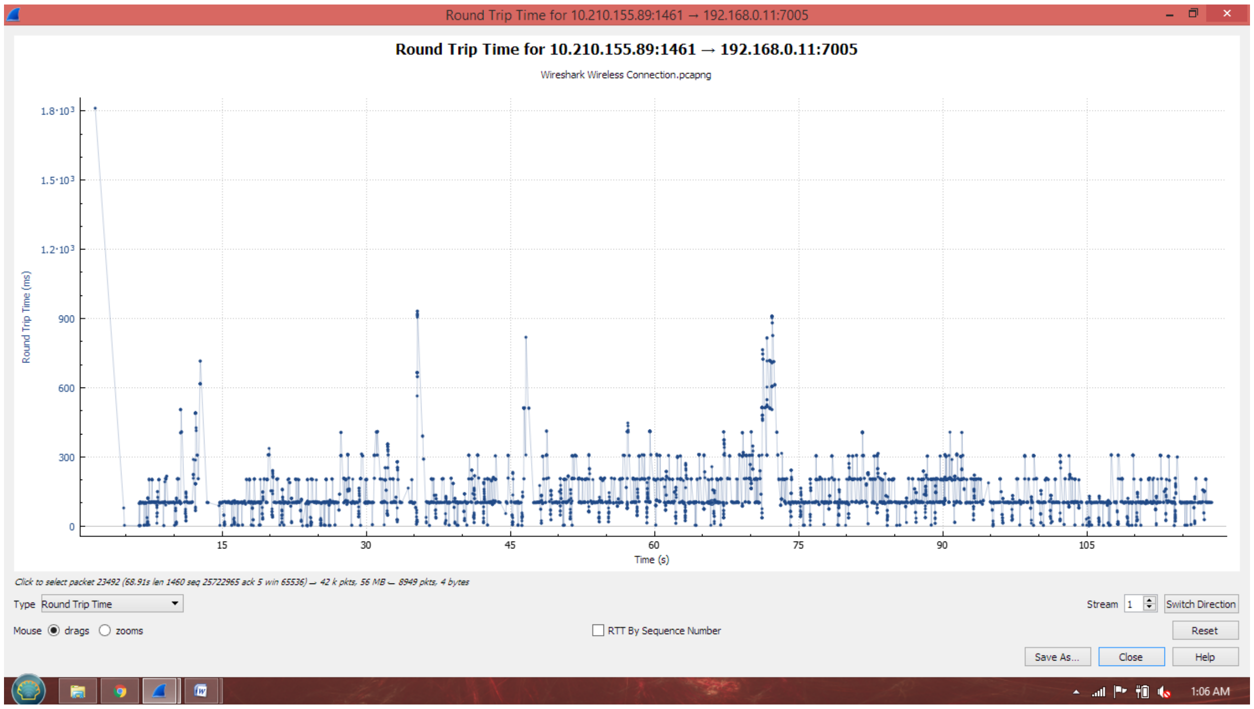

Sequence Numbers (Stevens) for 192.168.0.12:42130 → 192.168.0.11:7005
Wireshark Wired Connection.pcapng
Sequence Number (B)
Time (s)
Hover over the graph for details. — 1243 pkts, 55 MB — 4535 pkts, 4 bytes
Type Time / Sequence (Stevens)
Mouse drags zooms
Stream 0
Switch Direction
Reset
Save As...
Close
Help
9:32 PM

Interestingly, what can be seen is the initial connection to the server before the SEND command is sent. Once the server responds with a confirmation for sending, the data is sent. Slow start begins just right after the 9 second mark, and then from there the graph shows that the TCP progresses fairly linearly with some breaks in between that could have been from packets being lost. Although not quite as expected out of TCP as it could be, there was not much TCP congestion avoidance that was utilized.

Wireshark Wireless Connection TCP (Client-side)

Note: this section was done using the TCP server + client executables from assignment 1 in Linux as the server on a lab machine and a windows laptop as the client. Also the graphs being used are from using Wireshark 2.4.2

1.) **Plot a TCP Round Trip Time Graph and provide a clear and detailed explanation of what the graph is indicating.**



Click to select packet 23492 (68.91s len 1460 seq 25722965 ack 5 win 65536) → 42 k pkts, 56 MB ← 8949 pkts, 4 bytes

Type: Round Trip Time
Stream: 1
Mouse: (●) drags ( ) zooms
☐ RTT By Sequence Number

The graph above shows an initial start which could would have been from initiating a connection to the server and sending the command to send a text file. Following this begins a visible trend of upwards and downwards spikes. This could be due to the wireless network having a less stable connection so packets would have to be in queue on the router. The more "solid" horizontal line could be where the connection has the upper limit of traffic. The small spikes upwards tend to be a single dot, which could indicate that this is when the router begins to hit congestion in the traffic. The spikes downwards could be due to the wireless connection or router having a larger amount of buffering time for the TCP packet traffic. The larger spikes could be from packets being lost or an abnormally large set of data being sent. As this is wireless the traffic is more erratic, but remains somewhat consistent in trend.

2) **Select a set of 6 to 10 segments and analyze the difference between when each TCP segment was sent, and when its acknowledgement was received. Given your analysis, what is the RTT value for each of the segments you have selected? What is the EstimatedRTT value after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 249 for all subsequent segments.**

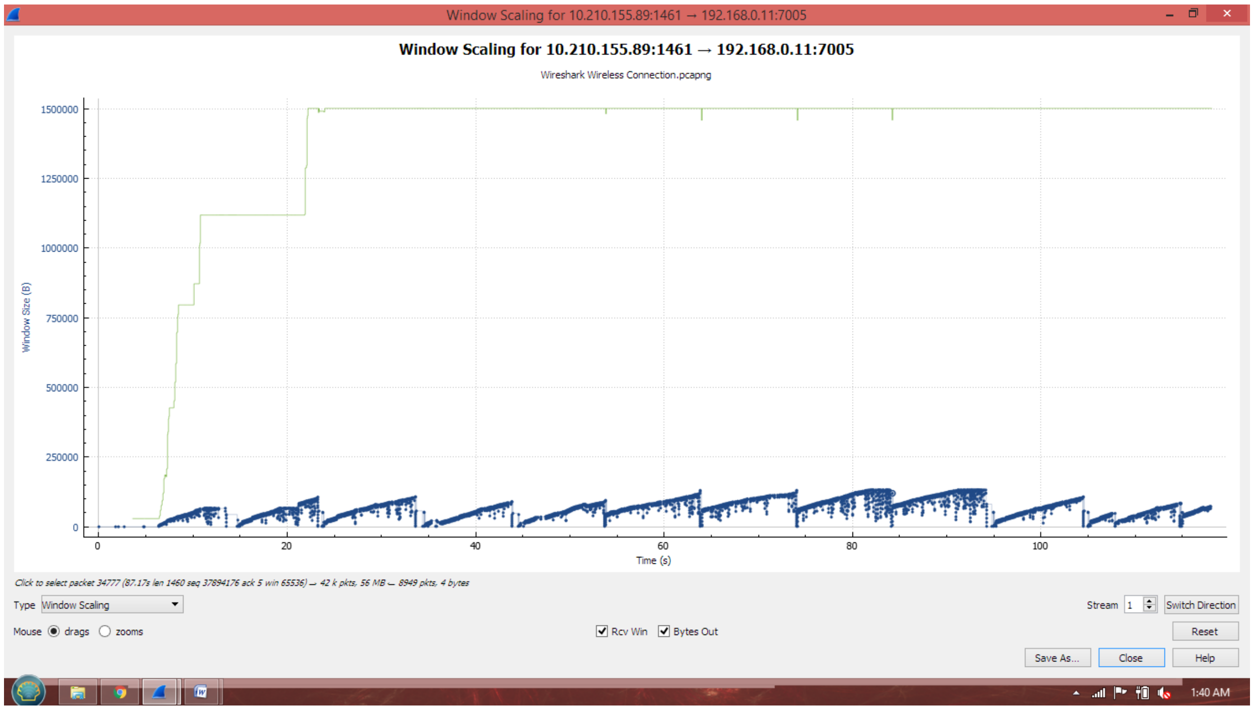
*EstRTT = (1 – α)*estimatedRTT + α*sampleRTT*
Packet 27) 0.001828000 *ACK = p28*
Packet 29) 0.001594000 *ACK = p31*
EstRTT = (1 – *0.125)* 0.001828000 + 0.125*0.001594000*
EstRTT = 0.0015995 + 0.00019925
EstRTT = 0.00179875

Packet 30) 0.001927000 *ACK = p34*
Packet 32) 0.098428000 *ACK = p36*
EstRTT = (1 – *0.125)* 0.001927000 + 0.125*0.098428000*

EstRTT = 0.001686125 + 0.012303125
EstRTT = 0.01398925

Packet 35) 0.098504000 *ACK = p38*
Packet 37) 0.002022000 *ACK = p42*
EstRTT = (1 – *0.125)* 0.098504000 + 0.125*0.002022000*
EstRTT = 0.086191 + 0.00025275
EstRTT = 0.08644375

3) What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender? Plot the Window size versus time for each of the data sets and comment on the results



Click to select packet 34777 (87.17s len 1460 seq 37894176 ack 5 win 65536) → 42 k pkts, 56 MB ← 8949 pkts, 4 bytes

Type: Window Scaling

Mouse: [✓ drags] zooms

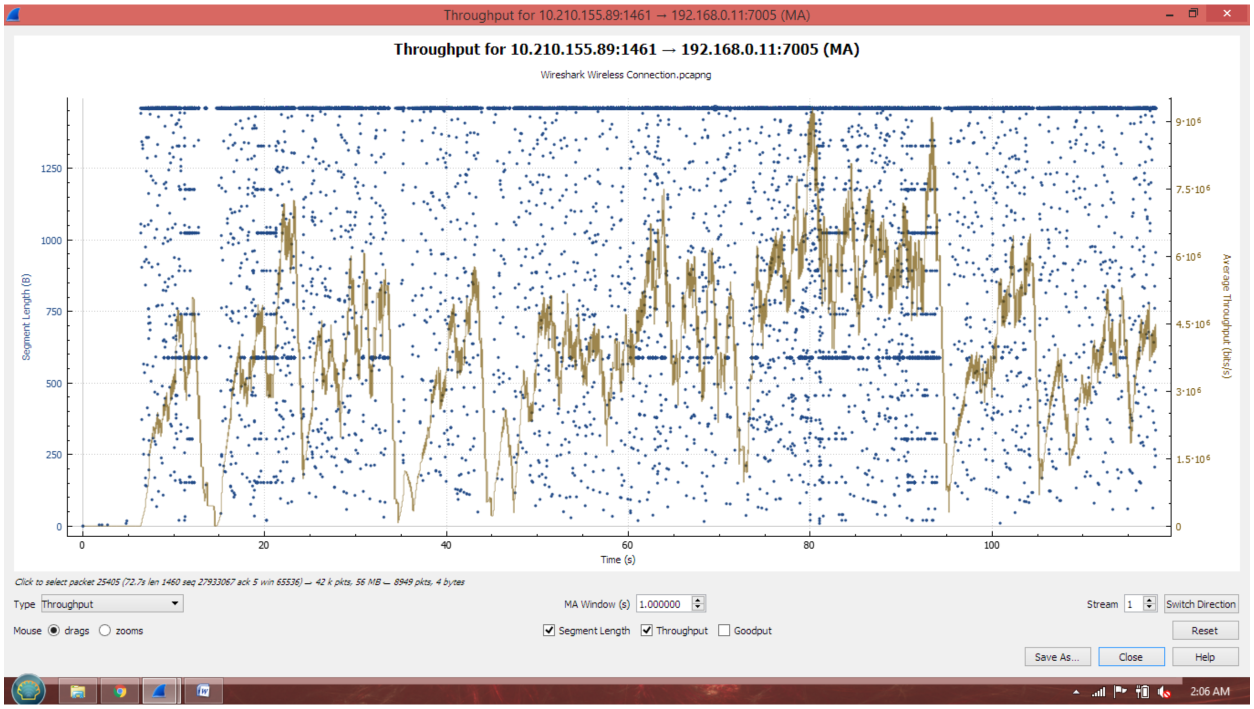
✓ Rcv Win ✓ Bytes Out

Stream: 1

The initial SYN, ACK segment has a window size of 29200. Lack of buffer space eventually does throttle the sender, as it can be seen with the window size suddenly dropping. As the buffer size changed to accommodate the data flow, it eventually has a loss from which the window size gets cut down significantly. This repeats several times leading to a saw-tooth effect which is shown in the graph

4) What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

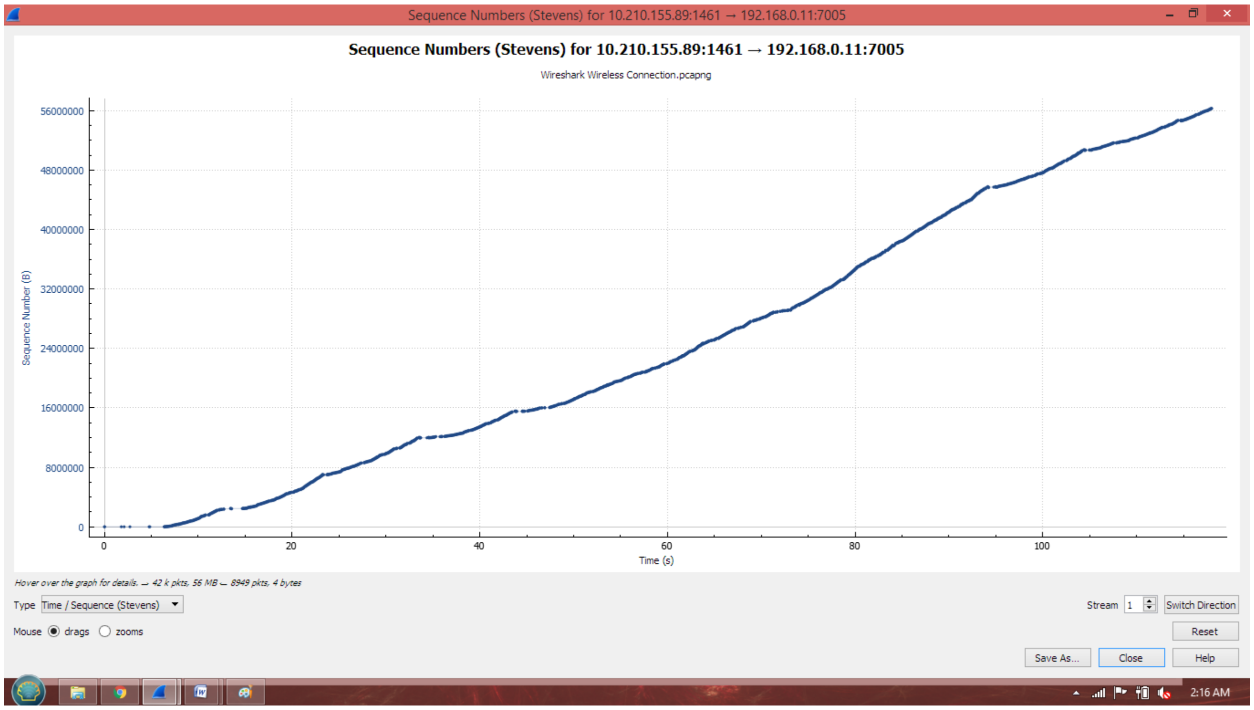
The size of the file sent was 55MB, or 55000kb. The total time to transfer the file was approximately 115 seconds. Using this data, the throughput was about 478.2608696 Kilobytes / second
Throughput = 55000/115



Click to select packet 25405 (72.7s len 1460 seq 27933067 ack 5 win 65536) — 42 k pkts, 56 MB — 8949 pkts, 4 bytes

Type Throughput
MA Window (s) 1.000000
Stream 1
Mouse (● drags) ○ zooms
☑ Segment Length ☑ Throughput ☐ Goodput

5) **Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the one machine to another. Identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we have discussed in lectures.**



Hover over the graph for details. — 42 k pkts, 56 MB — 8949 pkts, 4 bytes

Type Time / Sequence (Stevens)
Stream 1
Mouse (● drags) ○ zooms

At first the few small lone dots would have been from the initial connection to the server followed by the SEND command to the server with the file name. the Slow Start begins around the 6 second mark and quickly hits congestion avoidance, as a noticeable break in transmissions can be seen. From there a fairly linear shift occurs, with a small number of either congestion or packet loss occurs. This is almost what is idealized of TCP, but with small hiccups and the initial slow start to determine the linear behaviour.

Section B

1. **Read Jacobson's paper in detail. Clearly explain what he means by "The 'obvious' ways to implement a window-based transport protocol can result in exactly the wrong behaviour in response to network congestion."**

   When Jacobson mentioned that the 'obvious' ways to implement a window-based transport protocol can result in exactly the wrong behaviour in response to network congestion", he meant that majority of congestion problems lies within the transport protocol implementations. Formulating such protocols without error checking or simple solutions to problems could lead to bad performances and or the loss of data. Jacobson clearly discusses enhanced methods to avoid those issues through several sections of his paper including: the slow start algorithm targeting a connection to reach equilibrium, the RTT estimation for retransmitting packets, and congestion avoidance.

2. **Given what you have just read in this paper, provide a set of at least 3 guidelines that you would follow (as a software developer) in implementing an audio and video streaming application. Use the conclusions arrived at in the paper to justify your guideline.**

   Given what I have read, the 3 guidelines I would follow as a software developer are: a continuous connection to provide a seamless audio/video streaming experience for end-users, a buffer-free streaming experience, and a quicker response to streaming music/movies given a dynamic connection environment.

   a) Providing a seamless audio/video streaming service means having a connection stream that provides a 99.9999999% reliability rate for end-users. To follow this guideline, the algorithm must be able to retransmit packets at a fast rate if there is a packet delay or loss to keep the streaming service up and running without long buffer times. This guideline can be compared to the RTT estimator in which it estimates a retransmit timeout time for each new packet (not a fixed value) that is sent across the channel. This is very beneficial when working with a heavy load of packet transfers and it minimizes the congestion if packets are being delayed.

   b) Providing a buffer-free streaming experience is a means to prevent a streaming session from being choppy (buffering) to the point of a loss of connection. To prevent this from happening, I would implement a solution that automatically/dynamically changes the quality of the video/song if the connection is too weak to work with a heavy load. By reducing the quality of the stream, it can present a smoother flow to prevent the loss of packets which can lead to the problem of "loss of connection". This guideline can be compared to the congestion avoidance presented in Jacobson's paper in which the window size is reduced when the congestion variable is instigated and then increased when the packets are flowing more seamlessly (central flow of acks with minimal packet loss).

c) Ensuring a quicker transition to a good music/video stream when the service is just starting up means preventing slow buffering speeds when the connection is at its strongest. To prevent this from happening, I would implement a solution that load the media up bit by bit or in lower quality and once a sign of no packet loss is presented given the amount of resources available, and then slowly start to increase the media quality (more packets being sent across the channel) once the connection is recognized as a strong source. This guideline can be compared to the slow start algorithm discussed in Jacobson's paper in which few packets are transmitted at the start of the connection or when there's an interruption in the flow as in packet delays or losses. Once there is a smooth flow, the algorithm then injects more packets into the window at a growing rate to match the resources available to reach an equilibrium state.

Section C

P14: Consider a reliable data transfer protocol that uses only negative acknowledgements. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

In this question, we will be referencing the packets through sequence numbers as in: pkt0 → ack0 and pkt1 → ack1. In this scenario where the sender only sends data infrequently, it wouldn't be feasible to use a NAK-only protocol. In an example scenario: pkt0 gets sent over and is either delayed or lost in getting to the receiver. Since there are no ack, the sender proceeds to send over pkt1 and is received by the receiver. The only way the receiver can send a NAK (realize that pkt0 was not sent) is only when it receives pkt1 from the sender. This is when it realizes that pkt0 was not received initially, and from there can send a NACK0. Since the data is sent infrequently the time it takes for the receiver to detect this is far too long which makes this protocol inefficient.

Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

In this scenario where the sender sends data frequently, it would be feasible to use a NAK-only protocol. Since there are more data packets being sent over at a small time window, the receiver can realize a loss of packet much quicker since packets are constantly being sent over.

P46: Consider that only a single TCP Reno connection uses one 10Mbps link which does not buffer any data. Suppose that this link is the only congested link between the sending and receiving hosts. Assume that the TCP sender has a huge file to send to the receiver and the receiver's receive buffer is much larger than the congestion window. We also make the following assumptions: each TCP

segment is 1,500 bytes; the two-way propagation delay of this connection is 150 msec; and this TCP connection is always in congestion avoidance phase, that is ignore slow start.

(a) What is the maximum window size (in segments) that this TCP connection can achieve?

Max Window Size (W) = ?
TCP segment = 1500 bytes
Link Capacity = 10 Mbps
RTT (2-way propogation) = 150 msec
8 bits in a byte
MSS = Maximum Segment Size = 1 byte = 8 bits
Conversion from milliseconds to seconds: 10^-3
Conversion from Megabytes/second to bits/second: 10^6

Formula
Window size * segment size * (Max segment size/RTT) = Link Capacity

Answer
W_{max} * 1500bytes * (8 bits/0.15 seconds) = 10 * 10^6
W_{max} *80 000 = 10 * 10^6
W_{max} = (10 * 10^6) / 80 000
W_{max} = 125 segments

(b) What is the average window size (in segments) and average throughput (in bps) of this TCP connection?

Average Window Size is ¾ W
W_{avg} = ¾(125)
W_{avg} = 93.75 segments

Average throughput is ¾W per RTT
AVG throughput = ¾ (W/RTT) bytes/sec
AVG throughput = (93.75 segments * 1500 bytes * 8 bits / 150 ms) * 10^-3
AVG throughput = 7.5 Mbps

(c) How long would it take for this TCP connection to reach its maximum window again after recovering from a packet loss?

W_{max} = 125 segments
Once congestion window reaches its max capacity, threshold gets reduced to ½ of its values before timeout.

Time to recover = ½ W * RTT in seconds
Time to recover = ½ (125) * 0.15 seconds
Time to recover = 9.375 seconds → 9.38 seconds to recover and reach W_{max} again.

P55: In this problem we investigate whether either UDP or TCP provides a degree of end-point authentication.

a. **Consider a server that receives a request within a UDP packet and responds to that request within a UDP packet (for example, as done by a DNS server). If a client with IP address X spoofs its address with address Y, where will the server send its response?**

   The server will send its server response to IP Address X as UDP doesn't involve formal handshakes (acknowledgements). This means that that the protocol does not follow proper authentication methods and would send to whatever address is specified in the packet header.

b. **Suppose a server receives a SYN with IP source address Y, and after responding with a SYNACK, receives an ACK with IP source address Y with the correct acknowledgment number. Assuming the server chooses a random initial sequence number and there is no "man-in-the-middle", can the server be certain that the client is indeed at Y(and not at some other address X that is spoofing Y)?**

   Yes, the server can be certain that the client is at source address Y because it can perform a check to make sure the acknowledgement number are correct and that it aligns with previous SYN and ACK messages.