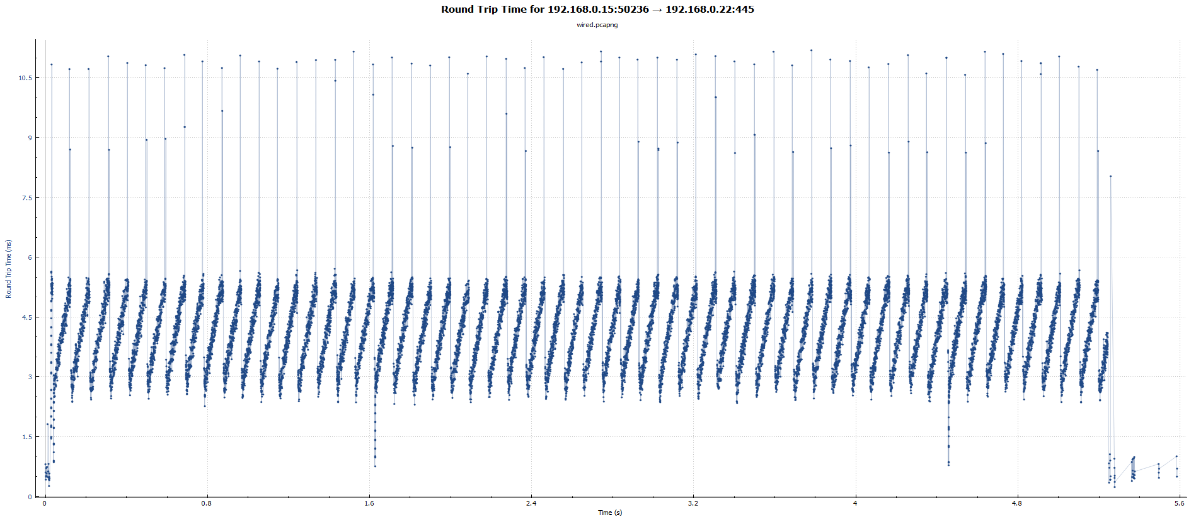
Peyman Tehrani Parsa A00922386 | Gaston Beaucage A00867938

Section A

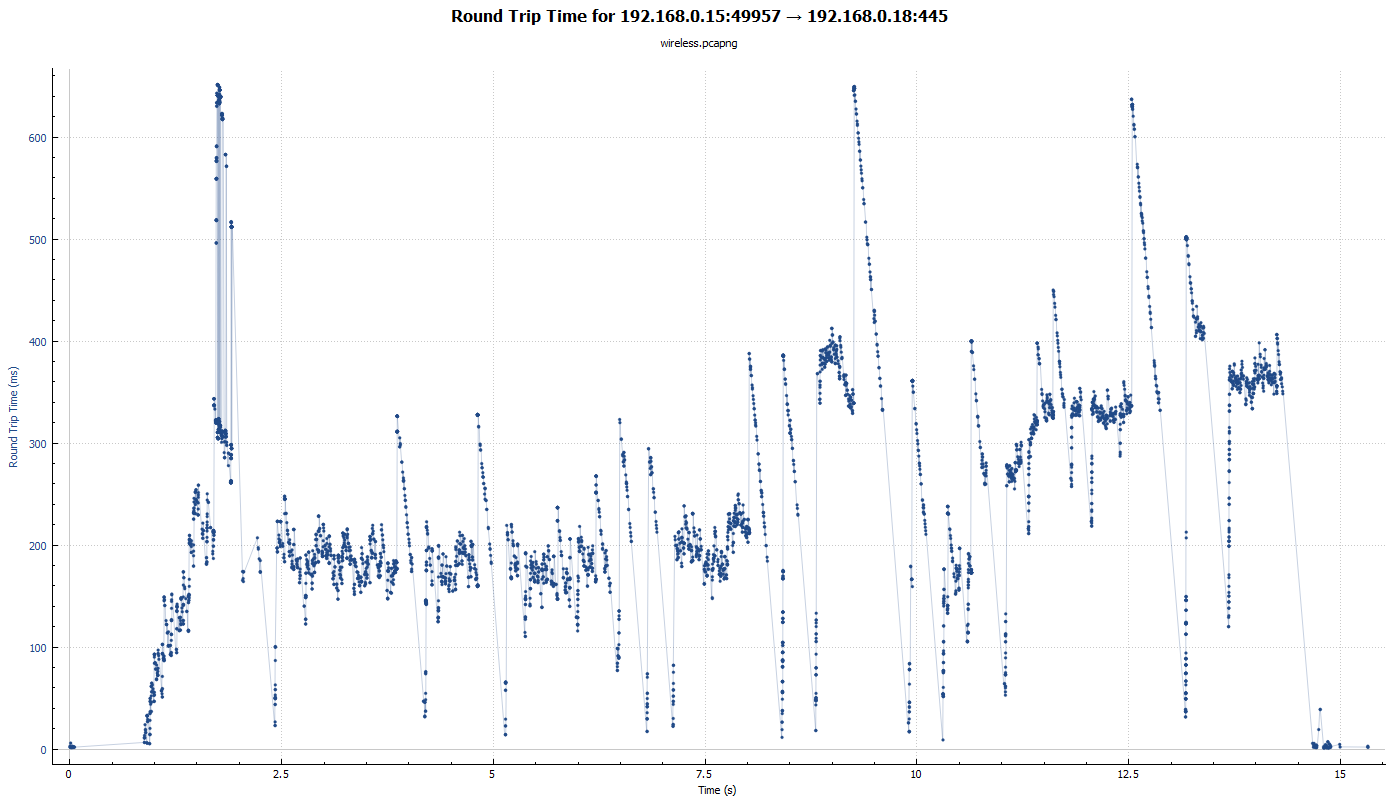
1. Plot a TCP Round Trip Time Graph and provide a clear and detailed explanation of what the graph is indicating.

1. **Wired**

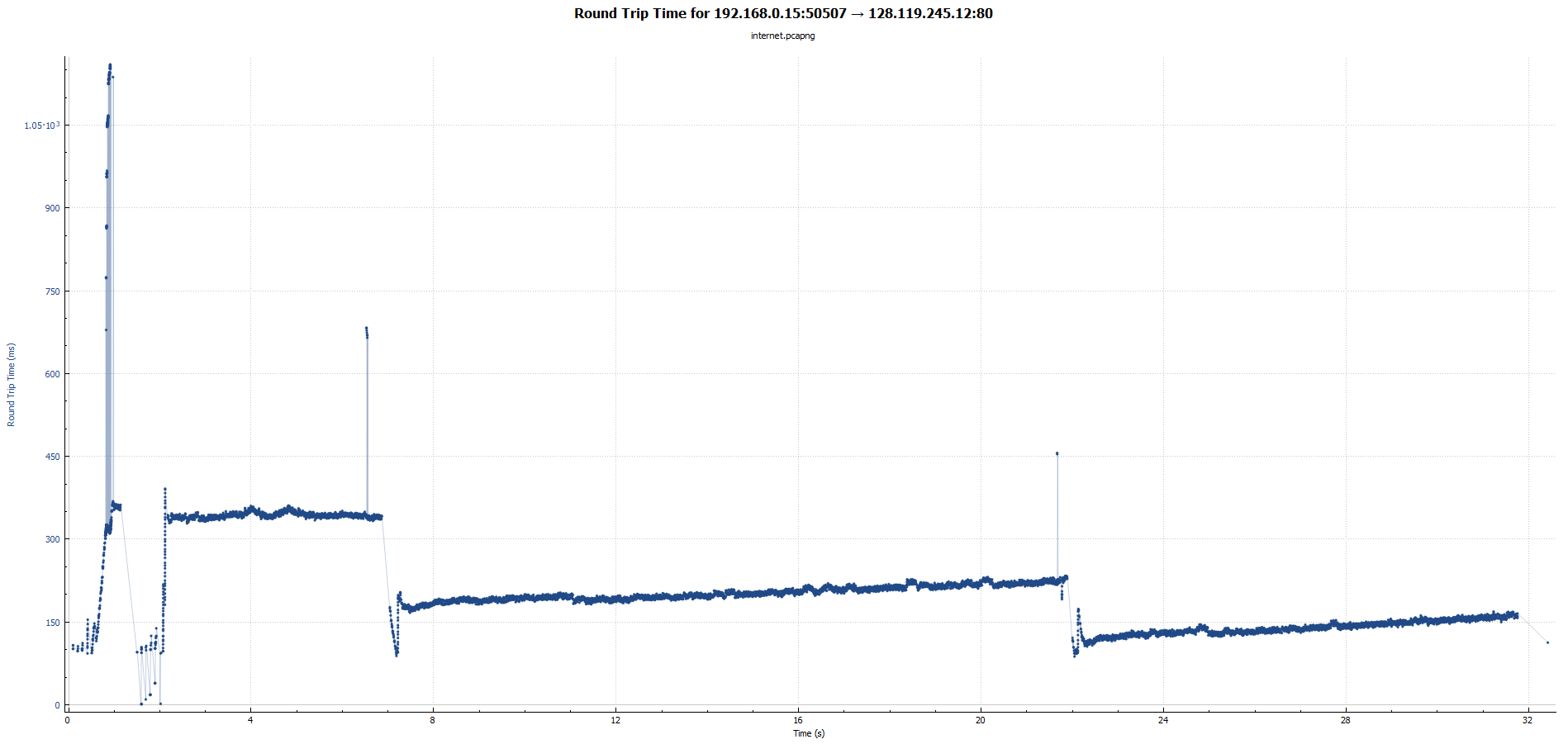
The graph shows AIMD saw tooth behavior. We can see linear growth until a packet is lost which causes the RTT to decrease by 1/2. We can see that around 57 losses happen.



1. **Wireless**

RTT starts with a slow start but because of lost packets due to noise on the line retransmissions are required increasing RTT and then the channel tries slow start again until noise strikes it down again

1. The RTT starts 100ms and increases exponentially using slow start until RTT is over one second causing packet losses(~0.912s) and a timeout(~1.6s). RTT returns to 100ms and exponentially increases with fast recovery until the dropped packets are resent and received. When RTT recovers at time ~2.17s to 50% of max, we start seeing linear growth with congestion avoidance. The pattern of packet loss, fast recovery, and congestion avoidance repeat at ~6.75 and ~21.9. The graph indicates that as the transfer continues the window size is decreasing.



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.

1. **Wired**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sequence number | Sent time (sec) | ACK time (sec) | RTT (sec) | Estimated RTT(sec) |
| Segment 1 | 7754 | 1.049775 | 1.051224 | 0.001449 | 0.001449 |
| Segment 2 | 13594 | 1.049837 | 1.051573 | 0.001736 | 0.875\*.001449+0.125\*0.001736= **0.001484875** |
| Segment 3 | 16514 | 1.049868 | 1.051867 | 0.001999 | 0.875\*0.001484875+0.125\*0.001999= **0.00154914062** |
| Segment 4 | 20894 | 1.049912 | 1.052119 | 0.002207 | 0.875\*0.00154914062+0.125\*0.002207= **0.00163137304** |
| Segment 5 | 23814 | 1.04994 | 1.052386 | 0.002446 | 0.875\*0.0016313730+0.125\*0.002446= **0.00173320137** |
| Segment 6 | 26734 | 1.052632 | 1.052632 | 0.002663 | 0.875\*0.**00173320137**+0.125\*0.002663= **0.00184942619** |

**Wireless**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sequence number | Sent time (sec) | ACK time (sec) | RTT (sec) | Estimated RTT(sec) |
| Segment 1 | 2349 | 1.247951 | 1.323431 | 0.07548 | **0.07548** |
| Segment 2 | 5269 | 1.247986 | 1.323432 | 0.075446 | 0.875\*0.07548+0.125\*0.075446= **0.07547575** |
| Segment 3 | 9649 | 1.248047 | 1.325323 | 0.077276 | 0.875\*0.0754757+0.125\*0.077276= **0.07570078125** |
| Segment 4 | 3199314 | 2.324291 | 2.473422 | 0.149131 | 0.875\*0.0757007+0.125\*0.149131= **0.08487955859** |
| Segment 5 | 3206614 | 2.324364 | 2.485154 | 0.16079 | 0.875\*0.08487955+0.125\*0.16079= **0.09436836376** |
| Segment 6 | 3225594 | 2.324557 | 2.498166 | 0.173609 | 0.875\*0.09436836+0.125\*0.1736= **0.1042734432** |

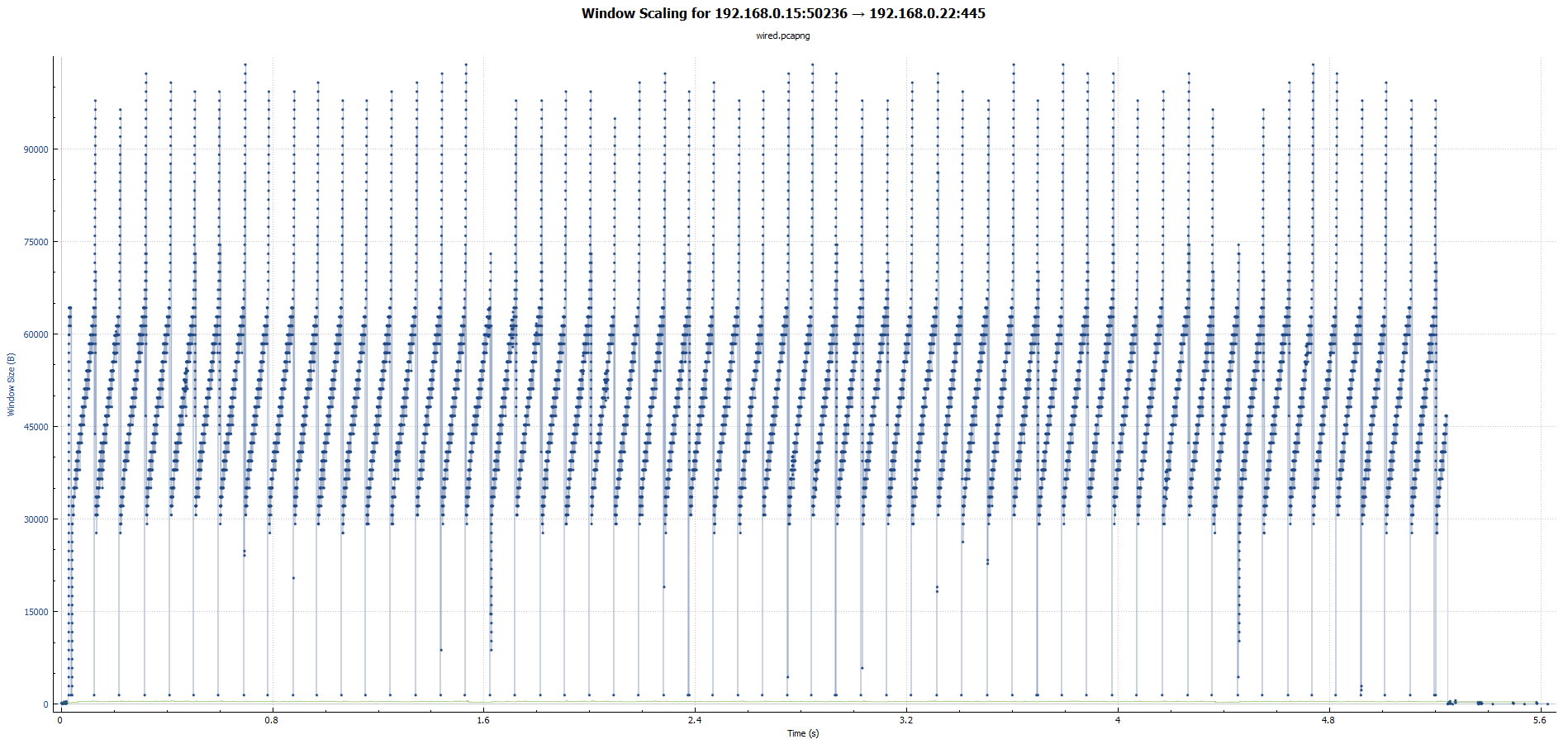


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sequence number | Sent time | ACK time | RTT | Estimated RTT(sec) |
| Segment 1 | 2921 | 0.752145s | 0.853444s | 0.101317s | **0.101317** |
| Segment 2 | 8761 | 0.752191s | 0.855495s | 0.103304s | 0.875\*0.101317+ 0.125\*0.103304=  **0.101565375** |
| Segment 3 | 10221 | 0.752206s | 0.859908s | 0.107702s | 0.875\*0.101565375+ 0.125\*0.107702=  **0.10233245312** |
| Segment 4 | 14601 | 0.75225s | 0.859908s | 0.107658s | 0.875\*0.10233245312+ 0.125\*0.107658=  **0.10299814648** |
| Segment 5 | 17521 | 0.853602s | 0.95089s | 0.097288s | 0.875\*0.10299814648+ 0.125\*0.097288=  **0.10228437817** |
| Segment 6 | 89061 | 0.961735s | 1.069756s | 0.108021s | 0.875\*0.10228437817+ 0.125\*0.108021=  **0.10300145589** |

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.

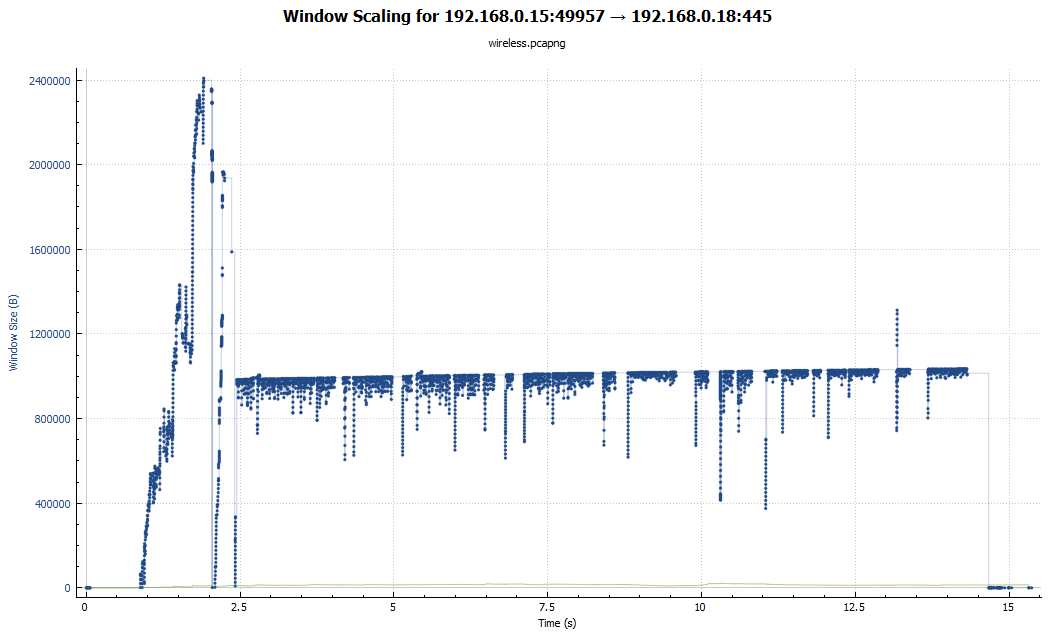
1. **Wired**

Min buffer space advertised is 2049 bytes, shown in the first ACK from the receiver and increases to a max of 103660 bytes. The transfer is showing saw tooth behavior. Constantly probing for bandwidth

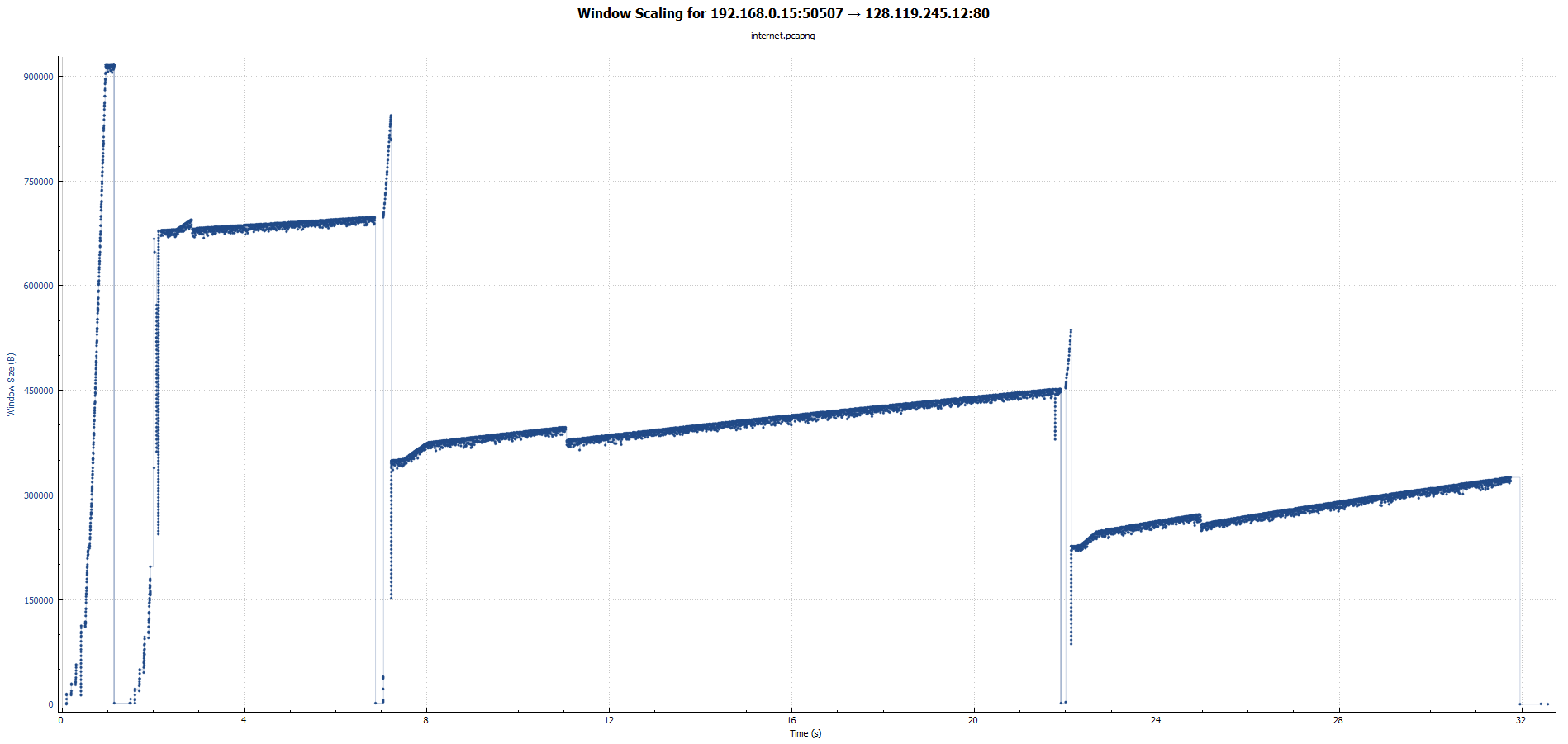


**Wireless**

Min buffer space advertised is 256 bytes, shown in the first ACK from the receiver and increases to a max of 13972 bytes. The sender is being throttled



1. Min buffer space advertised is 29200 bytes, shown in the first ACK from the receiver and increases to a max of 2660096 bytes. Sender is throttled because as soon as the receiver buffer space stops increasing, the throughput stabilizes.



4. What is the throughput (bytes transferred per unit time) for the TCP connection?

Explain how you calculated this value.

1. **Wired**

segment No.4 Time of [SYN,ACK] t1=1.02159 sec | S1=sequence number=1 byte

segment No.64365 Time of last ACK t2= 6.607535 sec | S2= 61907909 bytes

(S2-S1)/(t2-t1)= 11.08MB/s

**Wireless**

segment No.52 Time of [SYN,ACK] t1=1.03842sec | S1=sequence number=1 byte

segment No.46552 Time of last ACK t2= 16.3524sec | S2= 61918241 bytes

(S2-S1)/(t2-t1)= 4.043MB/s

1. segment No.13 Time of [SYN,ACK] t1=0.7521sec | S1=sequence number=1 byte

segment No.57566 Time of last ACK t2= 32.57498sec | S2= 61891219 bytes

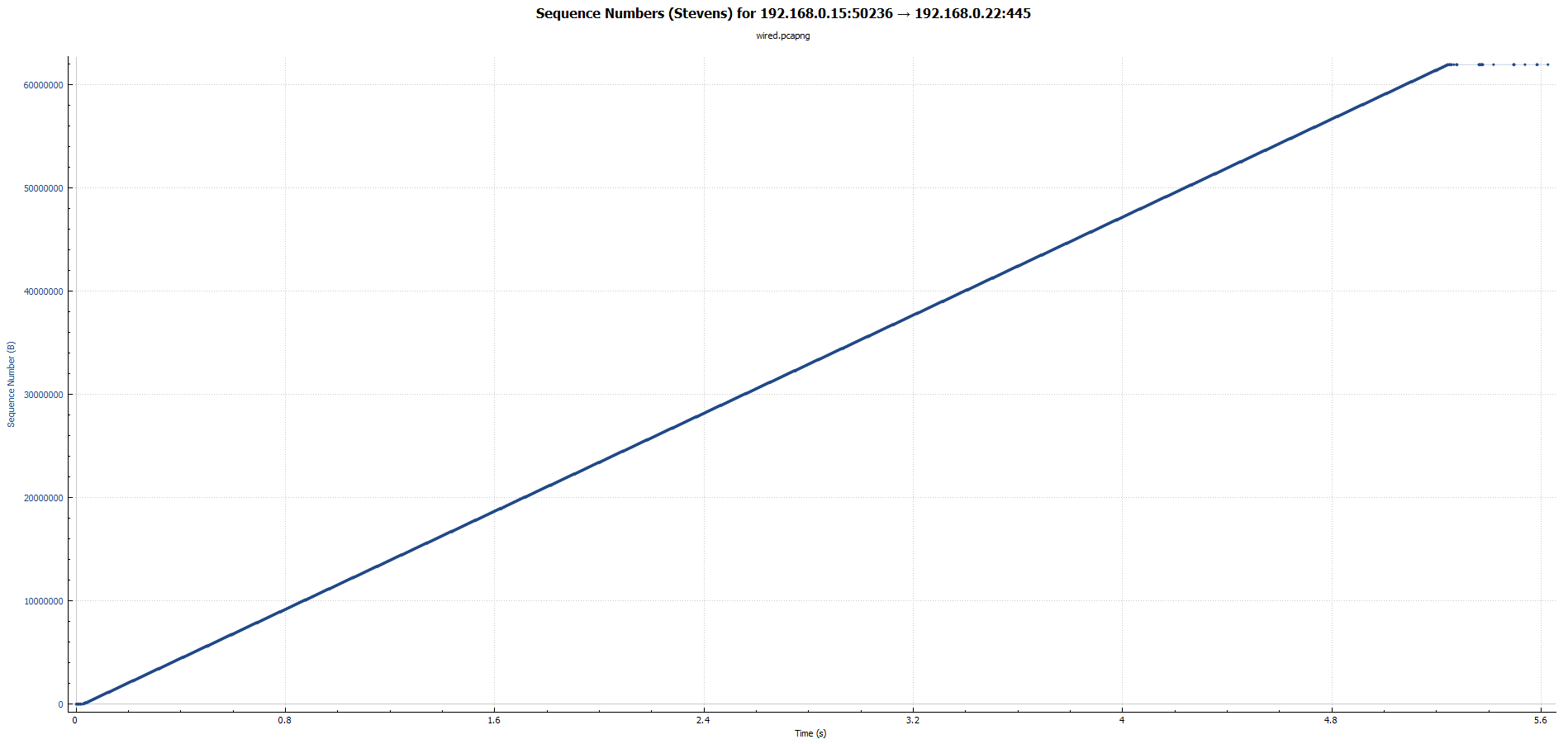
(S2-S1)/(t2-t1)= 1.854MB/s

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.

1. **Wired**

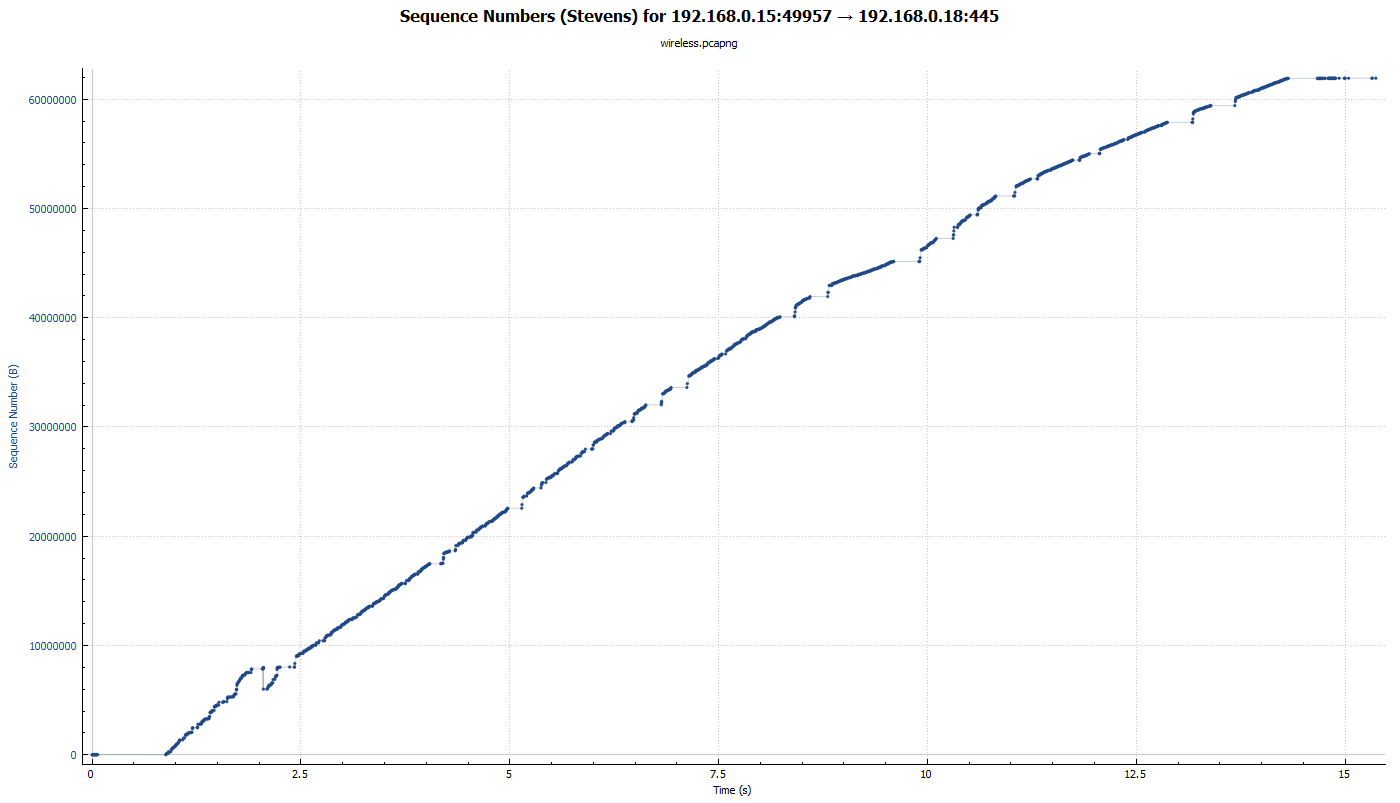
Slow start 1.05s-6.625s

Congestion avoidance never takes over



**Wireless**

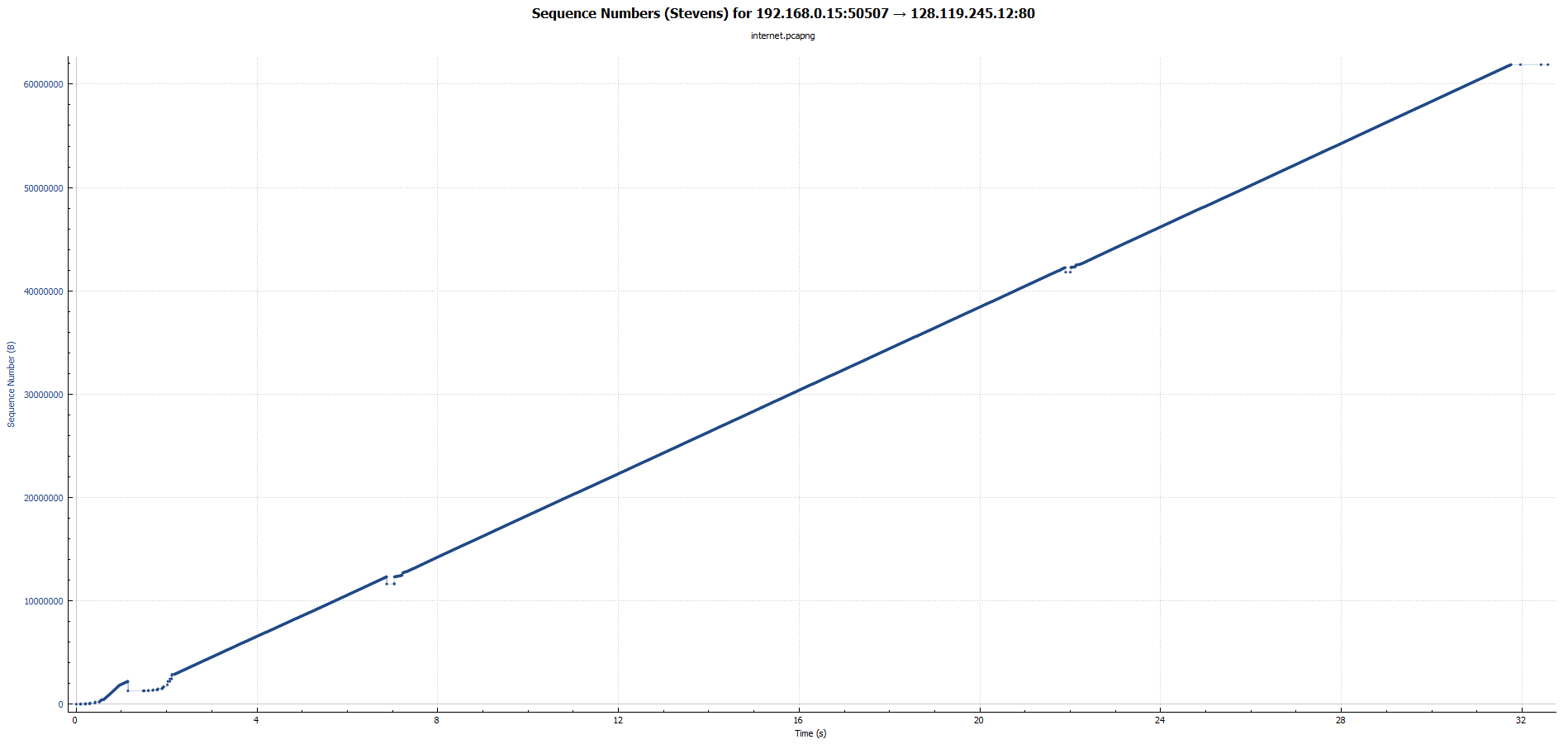
Slow start 1.039s-1.627s

Transfer never reaches fast recovery or congestion avoidance.

1. Slow start 0s-1.799266s

Fast Recovery 0-1.799266s-2.776s

Congestion avoidance 2.824s-7.524s



Section B:

1. What Jacobson means by the obvious ways to implement a window-based transport protocol can result in the exact opposite response to network congestion is that trying to immediately trying to maximize the bandwidth of a connection at the very beginning though in theory makes sense because you want to send as much information though as possible but when in practice can cause unnecessary amounts of retransitions. For example, Figure 3 demonstrates this principle. For this data the 54 to 58 KB data is sent five times.

Additionally, the next principle that Jacobson proposes is also a bit counter intuitive is that assuming the round trip time does vary and that you should throttle the data getting send with and exponential back off to prevent flooding the queue of gateways with packets. Because backups happen quickly!

Lastly the principle that you shouldn't change your window sizes too quickly in response to traffic because this will cause the system to oscillate widely and on average delivers poor throughput.

2. If I was implementing an audio and video streaming application 3 principles I would following my information of my streaming application would be as follows:

- I would start my UDP socket slowly and have an ACKing system in place so I know when the when packets aren't making it back by timeout. This would prevent me from saturating the system causing the need for retransitions.

* I would throttle video quality and reduce the quality of the video or audio if congestion occurs and reduce my window size respectively to prevent congestion from getting worse
* I would not assume round trip time was constant and try to buffer the media on the receiver in case the throughput suddenly dropped in the network do to congestion

Section C:

P14:

In a NAK protocol a NAK will be sent if the next packet in a sequence was sent and the current pack went missing in transit. This system that infrequently sends data would be bad because there would be a long delay until you figured out that you were out of sequence. As slow or slower than the rate that data is set. Whereas if you didn't receive an ACK in an ACK protocol within a timeout you would just resend the packet which would be faster. However, If data is sent more frequently this system becomes for efficient because data is NAKed as fast as data is transferred. It could be even better if there would be few errors because there would be less packets on the network controlling flow.

P46:

a. Tmax = Wmax (MSS / RTT)

therefore:

Wmax = Tmax \* (RTT / MSS)

Wmax = 10000000 bits/s \* (0.15 s / 1500bytes \* 8bit/byte)

Wmax = 125 bits

b. avg. window size (# in-flight bytes) is 3/4W

therefore

Wavg = 3/4Wmax

Wavg = 93.75 bits

Tavg = Wavg (MSS / RTT)

Tavg = 93.75 bits (1500bytes \* 8bit/byte/ 0.15s)

Tavg = 7500000 bits/s

c. the window decreases by half in the case for packet loss and increases it by one every RTT

therefore:

125 / 2 = half size

half size \* RTT = time to window max

62.5 \* 0.15s = 9.375s

P55:

a. The response will be sent to spoofed address Y there isn't anything protecting client from this.

b. You cannot be sure that the address is Y because the imposter could respond with the corresponding SYN send earlier.