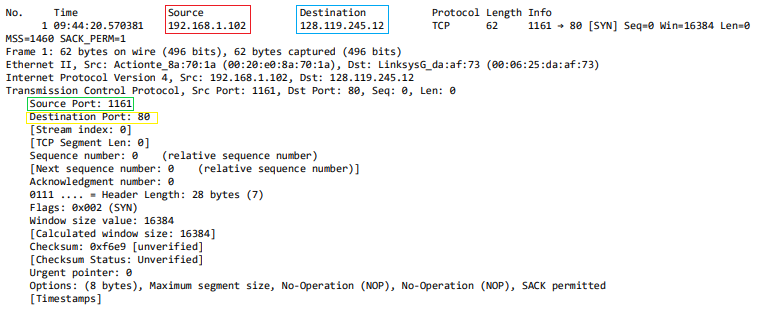
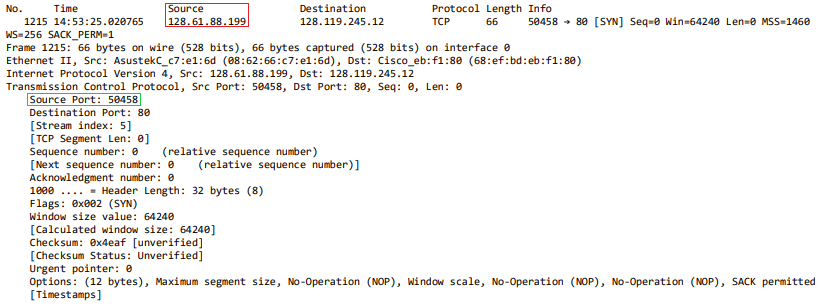
1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it’s probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the “details of the selected packet header window” (refer to Figure 2 in the “Getting Started with Wireshark” Lab if you’re uncertain about the Wireshark windows.
   1. Client/Source IP address: 192.168.1.102 (Marked in Red)
   2. Client/Source Port: 1161 (Marked in Green)
2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?
   1. Destination IP Address: 128.119.245.12 (Marked in Blue)
   2. Destination Port: 80 (Marked in Yellow)

Screen Shot for 1 and 2:



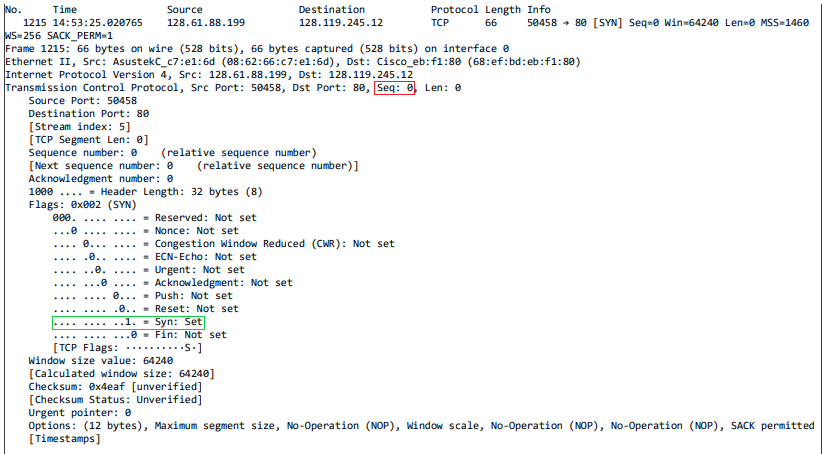
1. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?
   1. My Client/Sources IP address: 128.61.88.199 (Marked in Red)
   2. My Client/Sources Port: 50458 (Marked In Green)

Screen Shot for 3:



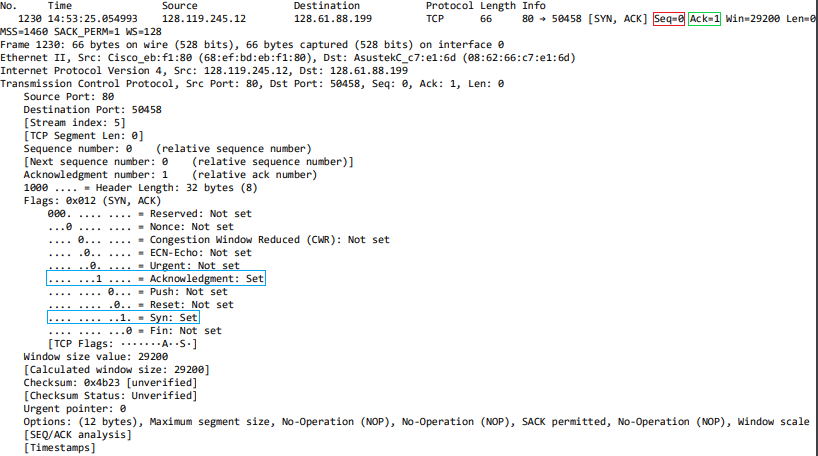
1. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?
   1. The Sequence Number of the TCP SYN Segment is: 0 (Marked In Red)
   2. Under the Flags part of the packet, there the Syn flag is set to 1 which means it is a SYN segment. (Marked in Green)

Screen Shot for 4:



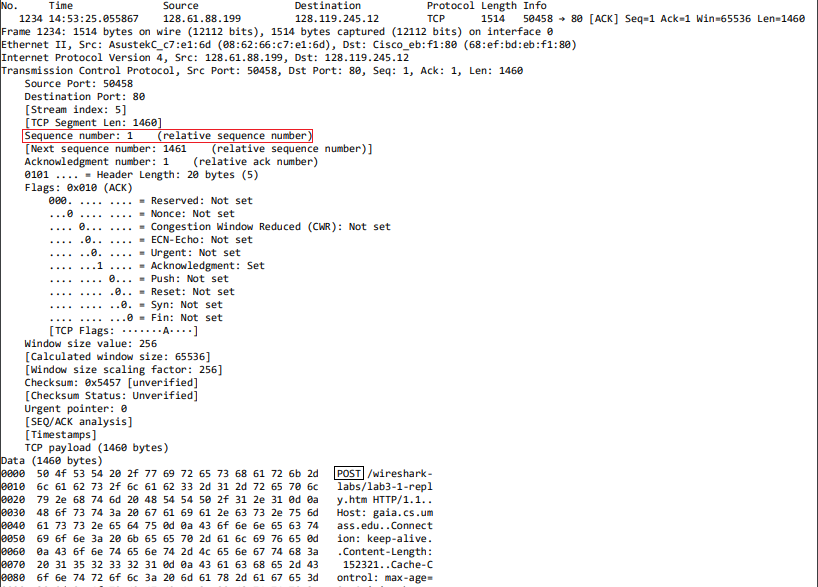
1. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?
   1. Sequence Number of SNACK segment: 0 (Marked In Red)
   2. Value of the Acknowledgement field: 1 (Marked in Green)
   3. The server adds 1 to the initial sequence number of SYN segment that is received from the client.
   4. The segment is identified as a SYNACK segment if both the SYN Set and Acknowledgement Set are 1. (Marked in Blue)

Screen Shot for 5:



1. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you’ll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a “POST” within its DATA field.
   1. The sequence number of the POST command is: 1 (Marked In Red)

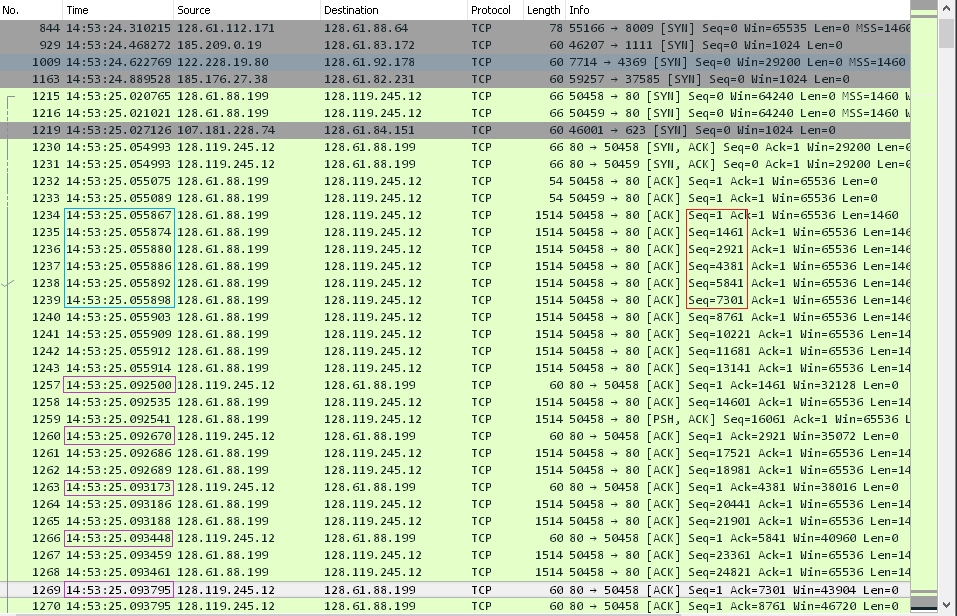
Screen Shot for 6:



1. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see Section 3.5.3, page 242 in text) 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 242 for all subsequent segments.
   1. The first 6 sequence numbers are: 1, 1461, 2921, 4381, 5841, 7301 (Marked in Red)
   2. Each Segment was sent at: 14:53:25.055867, 14:53:25.055874, 14:53:25.055880, 14:53:25.055886, 14:53:25.055892, 14:53:25.055898 (Marked in Blue)
   3. Each Segment was received at:(Could not find the receive time for Sequence No. 1) 14:53:25.092500, 14:53:25.092670, 14:53:25.093173, 14:53:25.093448, 14:53:25.093795 (Marked in Purple)

|  |  |  |  |
| --- | --- | --- | --- |
| Sequence Number: | Time Sent: | Time Received: | RTT: |
| 1 | 14:53:25.055867 | N/A | 0.0404  (Found RTT based on Graph) |
| 1461 | 14:53:25.055874 | 14:53:25.092500 | 0.03992825 |
| 2921 | 14:53:25.055880 | 14:53:25.092670 | 0.03959875 |
| 4381 | 14:53:25.055886 | 14:53:25.093173 | 0.040010875 |
| 5841 | 14:53:25.055892 | 14:53:25.093448 | 0.03969450 |
| 7301 | 14:53:25.055898 | 14:53:25.093795 | 0.040087125 |

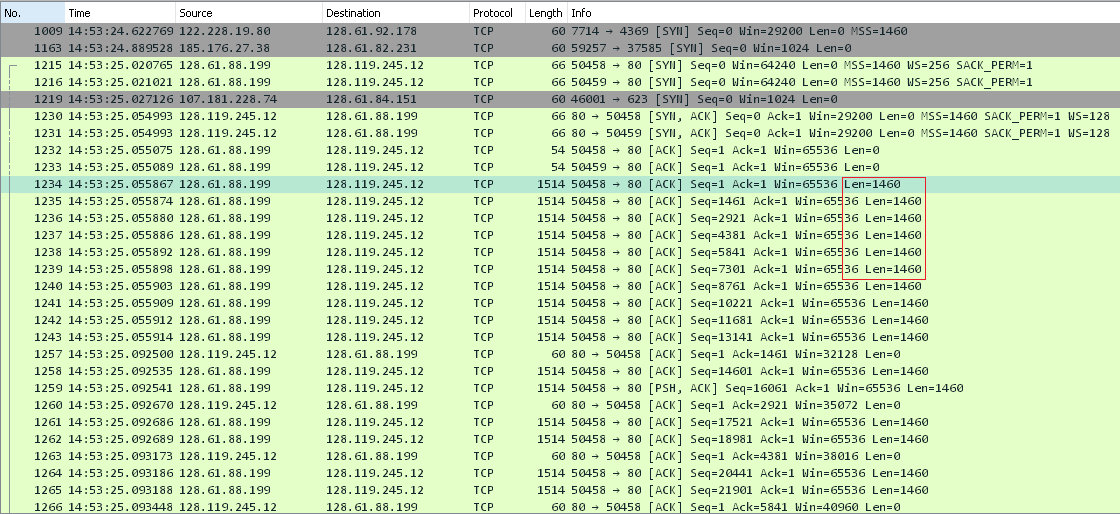
Screen Shot for 7:



1. What is the length of each of the first six TCP segments?
   1. Table:

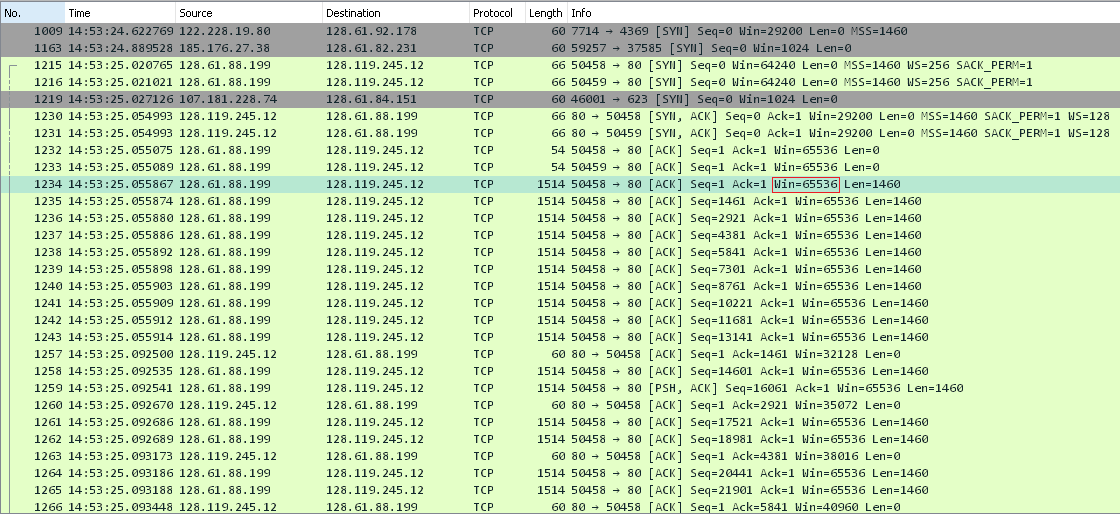
|  |  |  |
| --- | --- | --- |
| Packet No. | Sequence No. | TCP Segment Len:  (Marked in Red) |
| 1234 | 1 | 1460 |
| 1235 | 1464 | 1460 |
| 1236 | 2921 | 1460 |
| 1237 | 4381 | 1460 |
| 1238 | 5841 | 1460 |
| 1239 | 7201 | 1460 |

Screen Shot for 8:

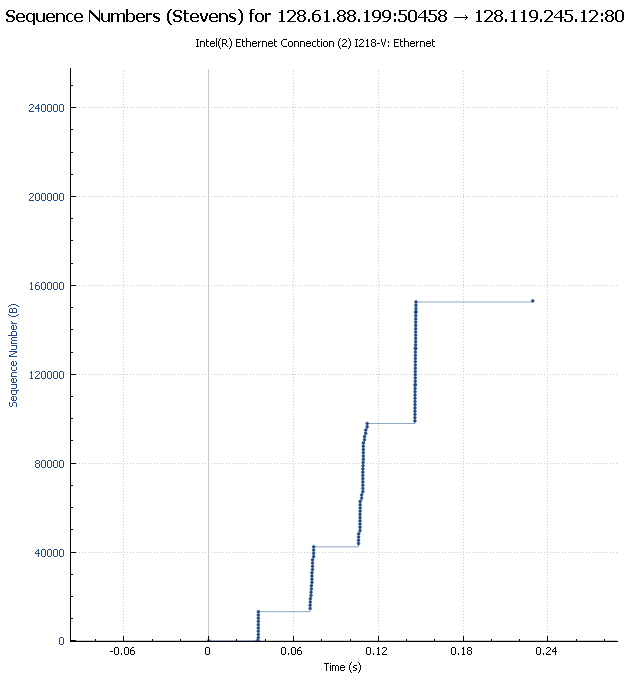


1. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
   1. The minimum amount of available buffer space advertised is: 65536 (Marked in Red)
   2. There is plenty of receiver buffer space so the sender is not throttled

Screen Shot of 9:

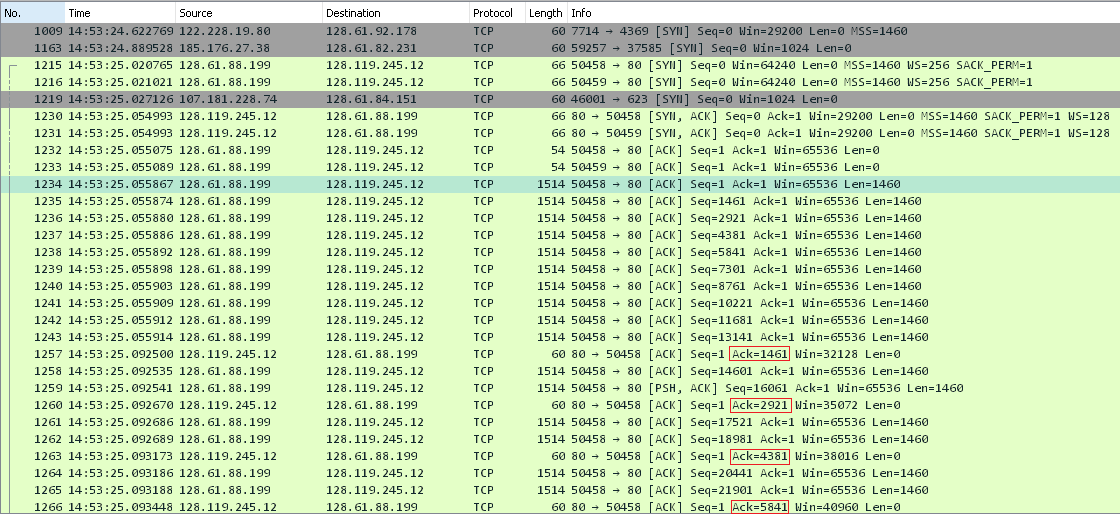


1. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
   1. No there were no retransmitted segments, you can determine this by looking at the figure below, where the same sequence no doesn’t appear at different times:



1. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 250 in the text).
   1. Based on the screen shot the receiver Acknowledges 1460 bytes at a time (Marked in Red)

Screen Shot of 11:



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

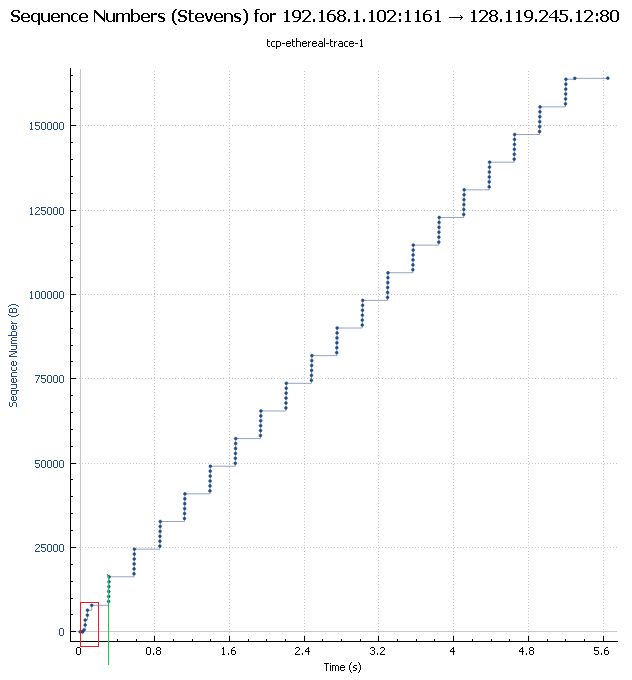
Screen Shot of 12:





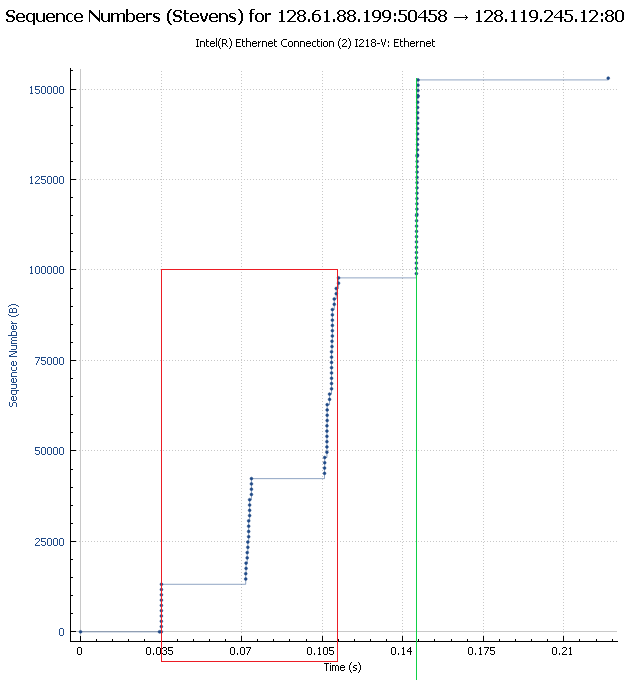
1. Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP’s slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we’ve studied in the text.
   1. The slow start phase looks to go from 0 sec to about 0.16 sec (Marked in Red)
   2. The congestion avoidance takes over at about about 0.32sec (Marked in Green)
   3. The graph is shows that the TCP Transmit window is not increase linearly instead it looks like we are transmitting a consistent 6 packets, this may be cause by the rate-limit of HTTP

Screen Shot of 13:



1. My Graph’s results:
   1. The slowstart phase begins at 0.035 secs and ends at about 0.11375 sec (Marked in Red)
   2. Congestion avoidance takes over at what looks to be about 0.14875 sec (Marked in Green)
   3. Because the buffer is so large it is difficult to see if measure data would behave ideally in that the window would increase linearly, if I were to make a guess based on the provided trace from Question 14 I would guess it would be have the same. It would normalize at an ideal number of packets and maintain that number as the size of the window.

Screen Shot of 14:



1. (P36) In Section 3.5.4, we saw that TCP waits until it has received three duplicate ACKs before performing a fast retransmit. Why do you think the TCP designers chose not to perform a fast retransmit after the first duplicate ACK for a segment is received?
   1. This was a design choice to help prevent redundant packet transfer.
      1. If a receiver get a packet with an unexpected sequence number (higher number) it will send the same ACK it sent previously, because the sender sends multiple segments at a time if one gets lost that would cause multiple duplicate ACKs to be sent back.
2. (P40) Consider Figure 3.58. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer
   1. Identify the intervals of time when TCP slow start is operating.
      1. You can see slow start is operating at [1,6] and [23, 26] because that is where you get exponential growth of the congestion window size
   2. Identify the intervals of time when TCP congestion avoidance is operating.
      1. You can see congestion avoidance operating at [6, 16] and [17, 22] because that is where you get linear congestion window size.
   3. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
      1. Segment Loss is detected by a triple duplicate ACK because if there was a timeout the window size would’ve dropped to 1.
   4. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
      1. Segment Loss is detected by timeout because the window size drops to 1.
   5. What is the initial value of ssthresh at the first transmission round?
      1. The window seems to be set to about 32 because that is where the congestion avoidance kicks in and the slow start phase ends.
   6. What is the value of ssthresh at the 18th transmission round?
      1. The window size is set to ½ of the window size when segment loss was detected by triple duplicate ACK, in this case at 16 we had a loss at a window size of roughly 45 so 18 we would start at a window size of 22 or 23.
   7. What is the value of ssthresh at the 24th transmission round?
      1. Because the window size is set to 1 when segment loss was detected by timeout at 22, the window size should be 2 or 3.
   8. During what transmission round is the 70th segment sent?

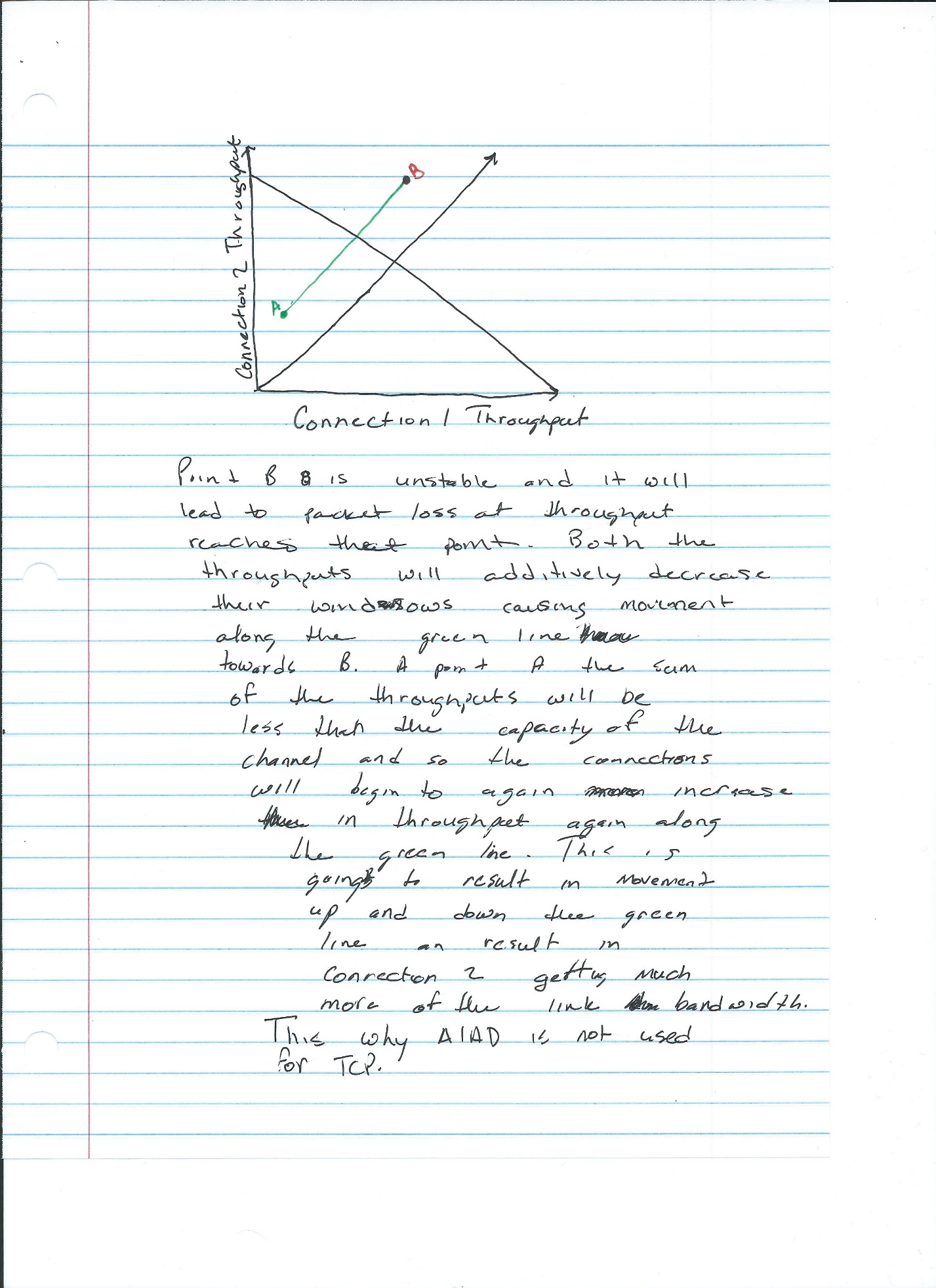
|  |  |
| --- | --- |
| Transmission Round: | Segments Sent: (approx..) |
| 1 | 1 |
| 2 | 2-3 |
| 3 | 4-7 |
| 4 | 8-15 |
| 5 | 16-31 |
| 6 | 32-63 |
| 7 | 64-96 |

* + 1. Based on the table it looks like the 70th segment was sent on the 7th transmission round.
  1. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?
     1. Assuming at the 26th transmission round the threshold is at 8 after that the threshold would be set to 4 because the same reason as f.
  2. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the ssthresh and the congestion window size at the 19th round?
     1. The threshold would be 21 and the congestion window size would be 1 at round 19
  3. Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?

|  |  |
| --- | --- |
| Transmission Round: | Packets sent: |
| 17 | 1 |
| 18 | 2 |
| 19 | 4 |
| 20 | 8 |
| 21 | 16 |
| 22 | 21 |

* + 1. A total of 52 packets would be sent.

1. (P41) Refer to Figure 3.55, which illustrates the convergence of TCP’s AIMD algorithm. Suppose that instead of a multiplicative decrease, TCP decreased the window size by a constant amount. Would the resulting AIAD algorithm converge to an equal share algorithm? Justify your answer using a diagram similar to Figure 3.55.



1. In our discussion of TCP congestion control in Section 3.7, we implicitly assumed that the TCP sender always had data to send. Consider now the case that the TCP sender sends a large amount of data and then goes idle (since it has no more data to send) at t1. TCP remains idle for a relatively long period of time and then wants to send more data at t2. What are the advantages and disadvantages of having TCP use the cwnd and ssthresh values from t1 when starting to send data at t2? What alternative would you recommend? Why?
   1. Advantage:
      1. Using earlier values of cwnd and ssthresh during t2, TCP would not have to deal with the congestion avoid and slow start and then get to the throughput that it reached during t1.
   2. Disadvantage:
      1. Using the previous cmnd and ssthresh has the inherent disadvantage that these values may not be accurate anymore.
         1. And example is where before t2 begins the path becomes more congested after t1, the sender would now be sending massive amounts of data over an already congested path, resulting in a performance hit.
   3. An alternative would be testing the previous cwnd and ssthresh and seeing how they perform, if the throughput is the same or similar to t1 then you can keep the values. If the performance is much worse then begin tweaking the values and testing performance. This method will cost more in terms of time but you no longer have to worry about the case of where the sender dumps a lot of data in an already congested path.