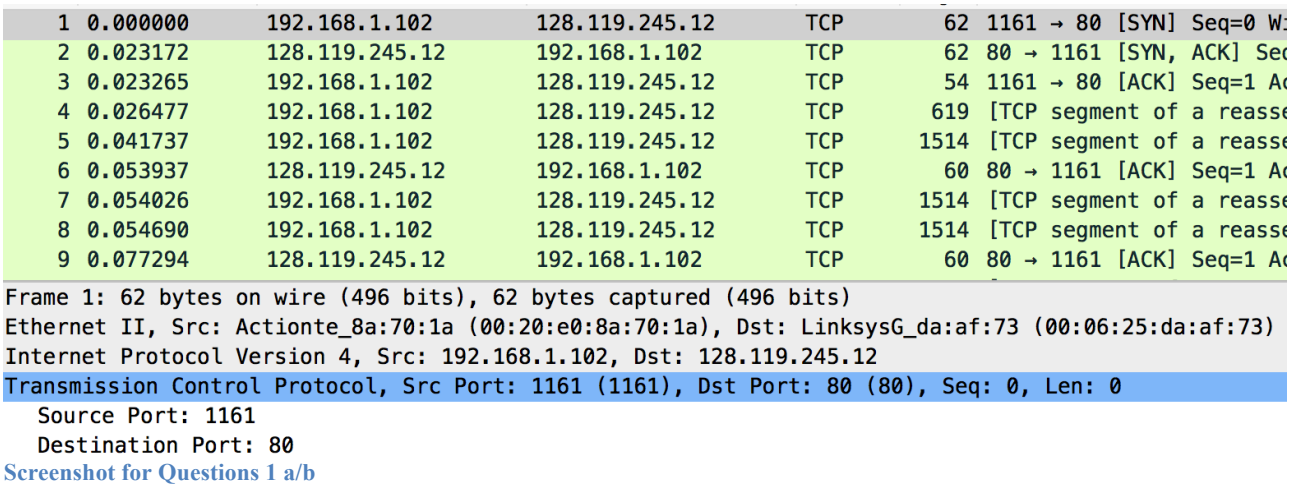
1.

a) 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.



Screenshot for Questions 1 a/b

Source TCP Port: 1161
Source IP Adress: 192.168.1.102

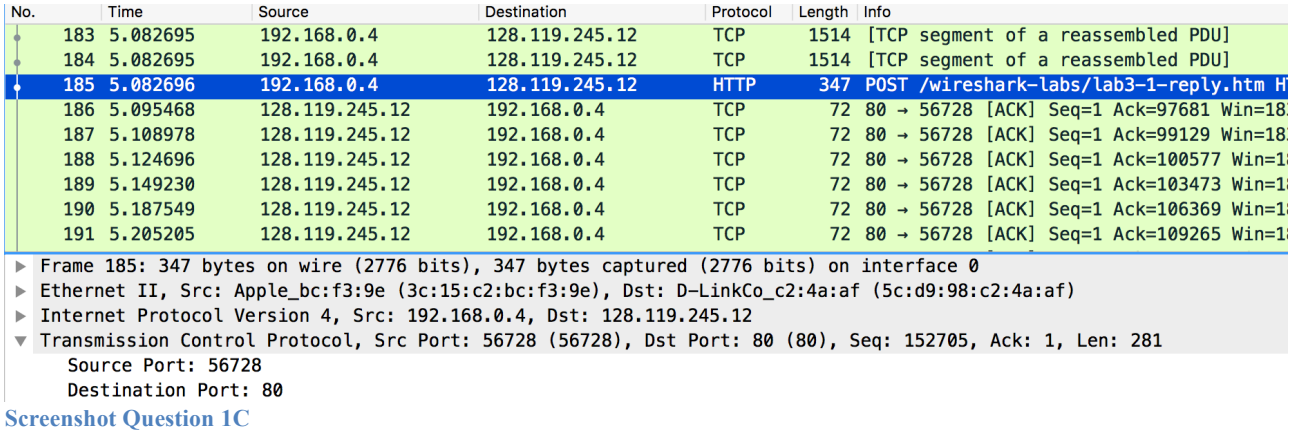
b) What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

IP Address of gai.cs.usmass.edu: 128.119.245.12
Port number receiving TCP segments: 80

If you have been able to create your own trace, answer the following question:

c) What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?

Source IP Address: 192.168.0.4
Source Port Number: 56728

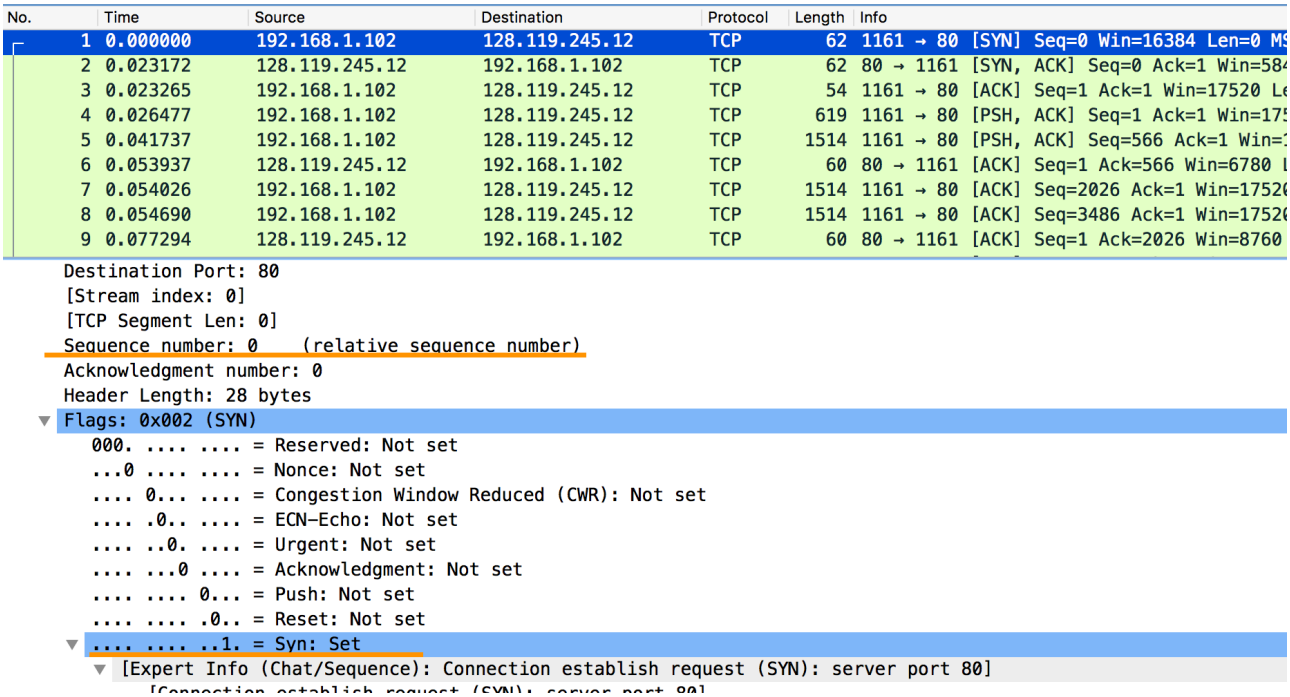


Screenshot Question 1C

Answer the following questions for the TCP segments:

2. *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?*

The sequence number is 0 and the SYN flag being set to 1 is what identifies this as a SYN segment, and that the sequence number is the initial sequence number.



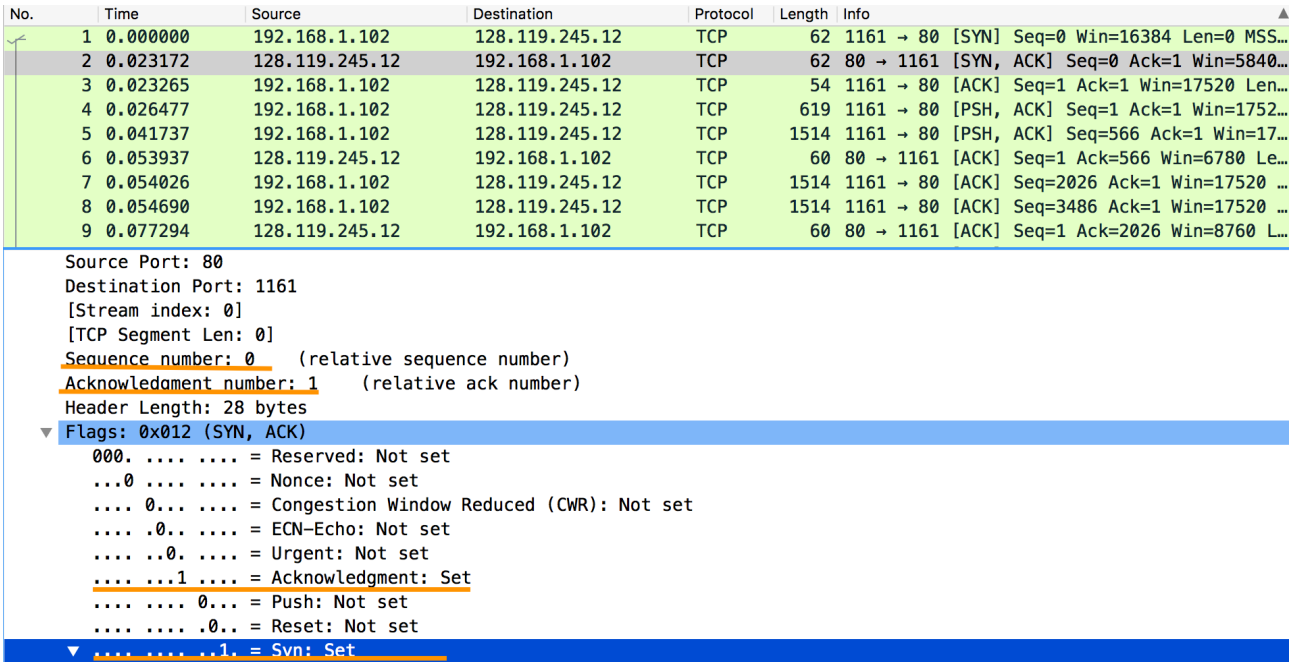
Screenshot Question 2

3. *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?*

The sequence number is 0.

The value of the Acknowledgement field is 1. After the initial SYN packet, the sequence number of the actual first data byte and the acknowledged number in the corresponding ACK are then this sequence number plus 1; to set the acknowledgement field, gaia.cs.umass.edu simply incremented the SYN value of the initial SYN packet by 1.

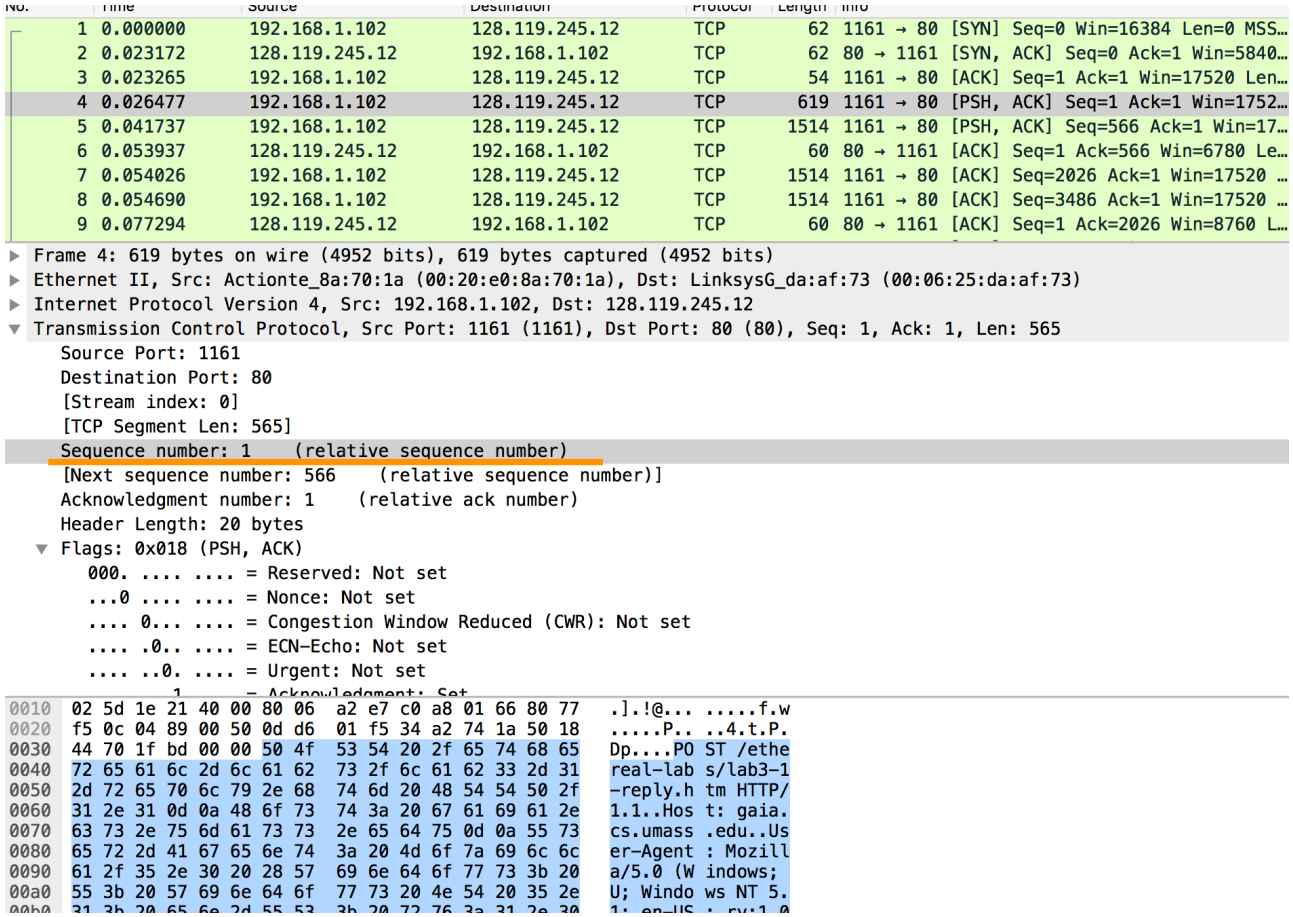
The ACK and SYN flags being set to 1 identify this segment as a SYNACK segment.



Screenshot Question 3

4. *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.*

The sequence number for the TCP segment containing the HTTP POST command has a **value of 1.**



Screenshot Question 4

5. *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 239 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 239 for all subsequent segments. Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: Statistics->TCP Stream Graph->Round Trip Time Graph.*

Breaking the question down:

a) What are the sequence numbers of the first six segments in the TCP connection (including the segement containing the HTTP POST)? At what time was each segment sent?

The first 6 TCP segments of the TCP connection are:

1. Seg. 4 (Sequence number 1), Sent at 0.026477

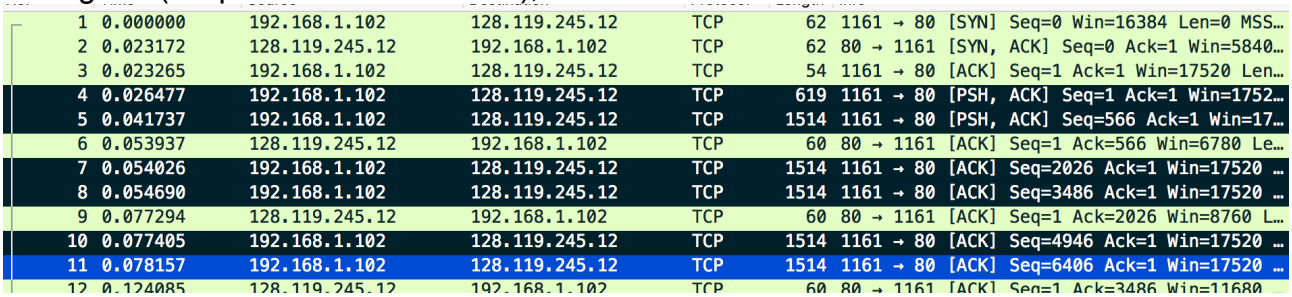
2. Seg. 5 (Sequence number 566), Sent at 0.041737

3. Seg. 7 (Sequence number 2026), Sent at 0.054026

4. Seg. 8 (Sequence number 3486), Sent at 0.054690

5. Seg. 10 (Sequence number 4946), Sent at 0.077405

6. Seg. 11 (Sequence number 6406), Sent at 0.078157

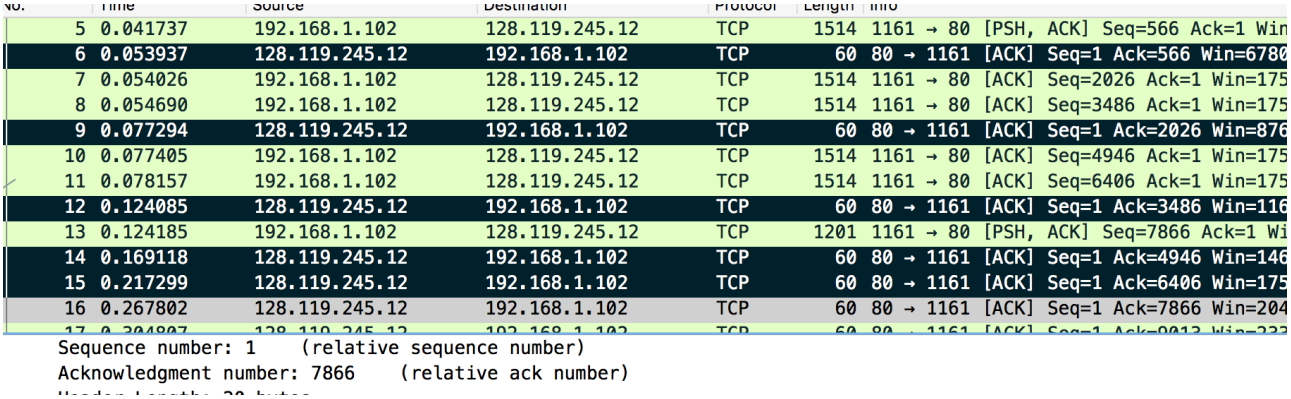


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.102	128.119.245.12	TCP	62	1161 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS…
2	0.023172	128.119.245.12	192.168.1.102	TCP	62	80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840…
3	0.023265	192.168.1.102	128.119.245.12	TCP	54	1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len…
4	0.026477	192.168.1.102	128.119.245.12	TCP	619	1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=1752…
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win=17…
6	0.053937	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Le…
7	0.054026	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=2026 Ack=1 Win=17520 …
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=3486 Ack=1 Win=17520 …
9	0.077294	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 L…
10	0.077405	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=4946 Ack=1 Win=17520 …
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=6406 Ack=1 Win=17520 …
12	0.124085	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 …

Screenshot Question 5a

b) When was the ACK for each segment received?

The first six ACKs:
1. Seg. 6 (received at 0.053937)
2. Seg. 9 (received at 0.077294)
3. Seg. 12 (received at 0. 0.124085)
4. Seg. 14 (received at 0.169118)
5. Seg. 15 (received at 0.217299)
6. Seg. 16 (received at 0.267802)



No.	Time	Source	Destination	Protocol	Length	Info
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win
6	0.053937	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780
7	0.054026	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=2026 Ack=1 Win=175
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=3486 Ack=1 Win=175
9	0.077294	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=2026 Win=876
10	0.077405	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=4946 Ack=1 Win=175
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=6406 Ack=1 Win=175
12	0.124085	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=3486 Win=116
13	0.124185	192.168.1.102	128.119.245.12	TCP	1201	1161 → 80 [PSH, ACK] Seq=7866 Ack=1 Wi
14	0.169118	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=4946 Win=146
15	0.217299	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=6406 Win=175
16	0.267802	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=7866 Win=204

Sequence number: 1 (relative sequence number)
Acknowledgment number: 7866 (relative ack number)

Screenshot Question 5b

c) 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?

Segment 1: 0.053937 - 0.026477 = **0.02746**

Segment 2: 0.077294 - 0.041737 = **0.035557**

Segment 3: 0.124085 - 0.054026 = **0.070059**

Segment 4: 0.169118 - 0.054690 = **0.11443**

Segment 5: 0.217299 - 0.077405 = **0.13989**

Segment 6: 0.267802 - 0.078157 = **0.18964**

d) What is the `EstimatedRTT` value (see Section 3.5.3, page 239 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 239 for all subsequent segments.

According to the book,

EstimatedRTT = 0.875 • EstimatedRTT + 0.125 • SampleRTT

After arrival of first ACK:
Assuming EstimatedRTT = RTT for first segment, Estimated RTT = **0.02746 second**.

After second ACK:
EstimatedRTT = (0.875 * 0.02746) + (0.125 * 0.035557) ≈ **0.0285 second**

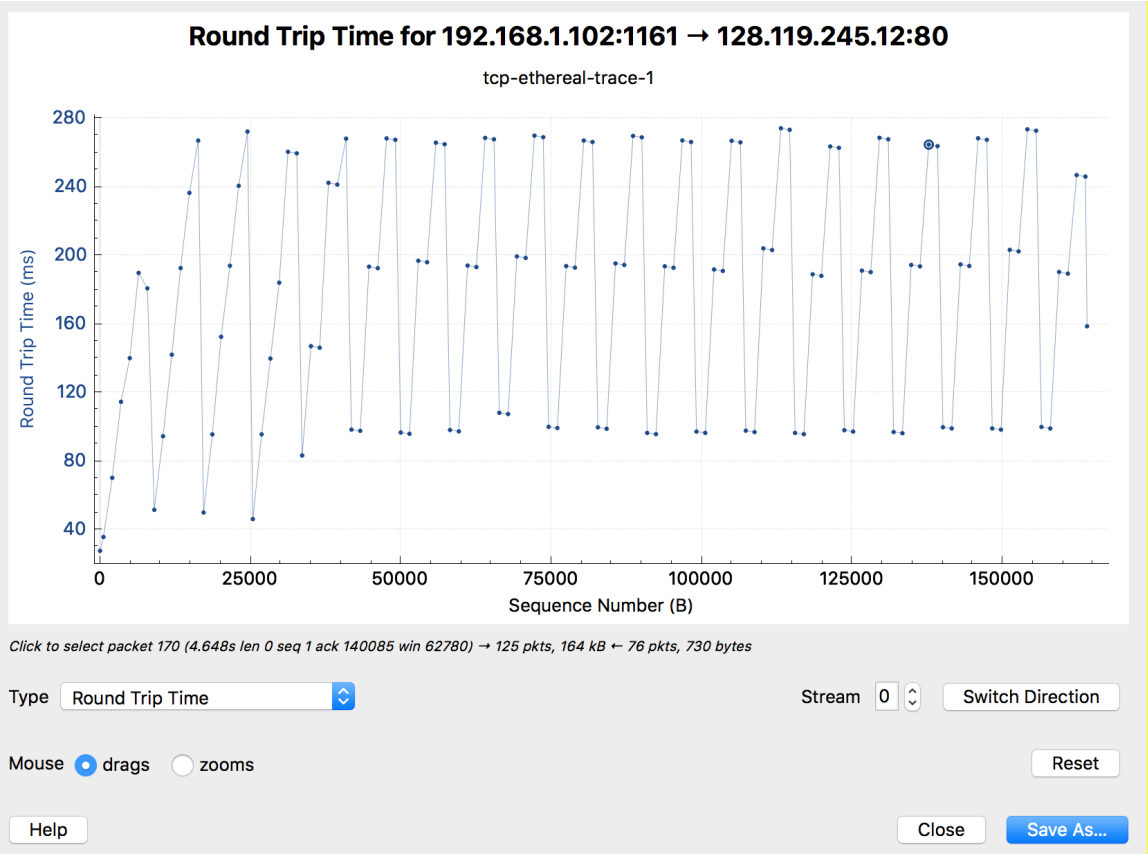
After third ACK:
EstimatedRTT = (0.875 * 0.0285 + (0.125 * 0.070059) ≈ **0.0337 second**

After fourth ACK:
EstimatedRTT =(0.875 * 0.0337)+ (0.125 * 0.11443) ≈ **0.0438 second**

After fifth ACK:
EstimatedRTT = (0.875 * 0.0438) + (0.125 * 0.13989) ≈ **0.0558 second**

After sixth ACK:
EstimatedRTT =(0.875 * 0.0558) + (0.125 * 0.18964) ≈ **0.0725 second**

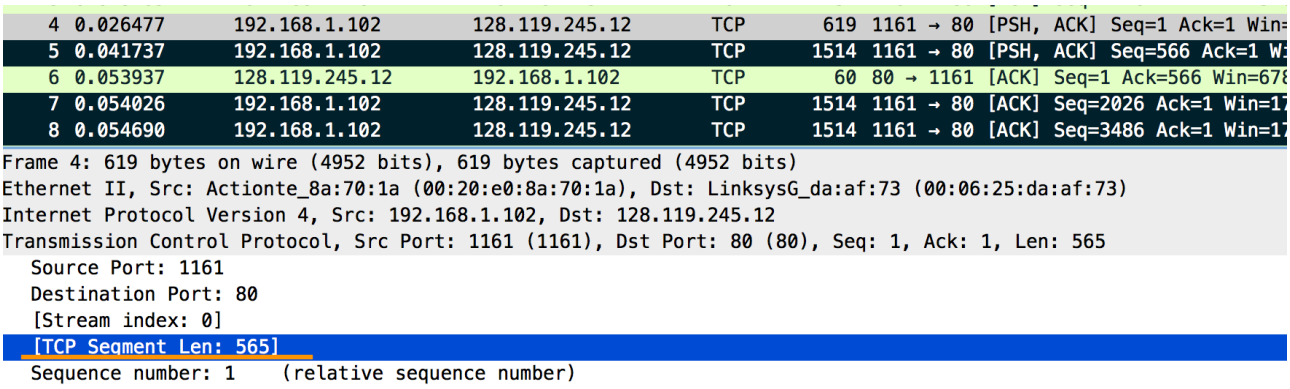
e) Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: Statistics->TCP Stream Graph->Round Trip Time Graph.:



Screenshot Note 5e

6. *What is the length of each of the first six TCP segments?*[4]

The length of the first segment is **565.**



Screenshot 1 Question 6

The lengths of the remaining 5 segments is **1460** for each.

```
    10 0.077405    192.168.1.102    128.119.245.12    TCP    1514 1161 → 80 [ACK] Seq=494
    11 0.078157    192.168.1.102    128.119.245.12    TCP    1514 1161 → 80 [ACK] Seq=640
    12 0.124085    128.119.245.12   192.168.1.102     TCP      60 80 → 1161 [ACK] Seq=1 A
    13 0.124185    192.168.1.102    128.119.245.12    TCP    1201 1161 → 80 [PSH, ACK] Se

Frame 11: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
Ethernet II, Src: Actionte_8a:70:1a (00:20:e0:8a:70:1a), Dst: LinksysG_da:af:73 (00:06:25:da:af:73)
Internet Protocol Version 4, Src: 192.168.1.102, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 1161 (1161), Dst Port: 80 (80), Seq: 6406, Ack: 1, Len: 14
   Source Port: 1161
   Destination Port: 80
   [Stream index: 0]
   [TCP Segment Len: 1460]
```

Screenshot 2 Question 6

7. a) What is the minimum amount of available buffer space advertised at the received for the entire trace?

The minimum window size advertised by the receiver (flow control) is **5840** bytes.

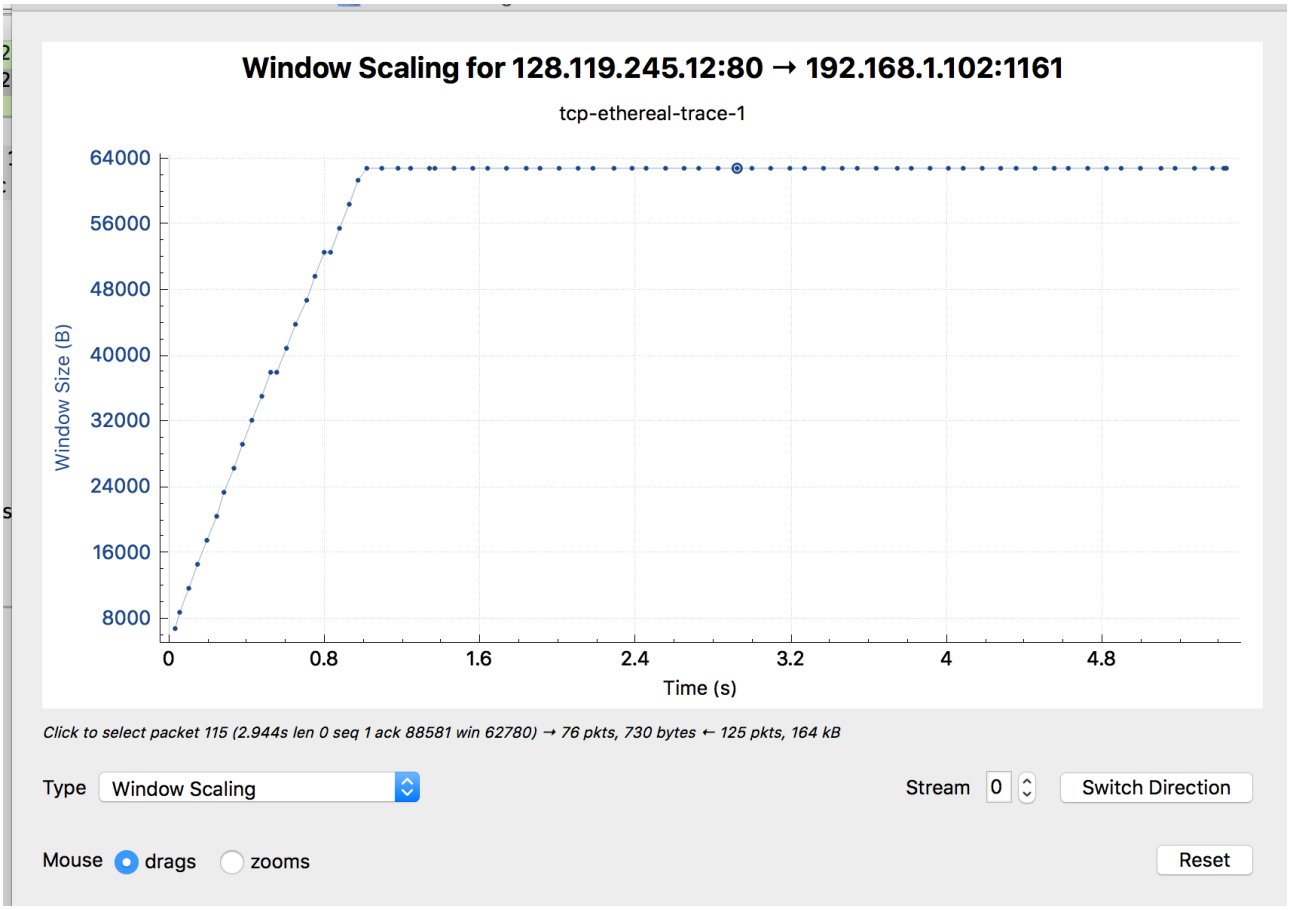
Screenshot:

```
No.  Time       Source           Destination      Protocol  Length  Info
     1 0.000000 192.168.1.102    128.119.245.12   TCP         62    1161 → 80 [SYN] Seq=0 Win=163
     2 0.023172 128.119.245.12   192.168.1.102    TCP         62    80 → 1161 [SYN, ACK] Seq=0 Ac
     3 0.023265 192.168.1.102    128.119.245.12   TCP         54    1161 → 80 [ACK] Seq=1 Ack=1 W
   Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.1.102
▼ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 1161 (1161), Seq: 0, Ack: 1, Len: 0
   Source Port: 80
   Destination Port: 1161
   [Stream index: 0]
   [TCP Segment Len: 0]
   Sequence number: 0    (relative sequence number)
   Acknowledgment number: 1    (relative ack number)
   Header Length: 28 bytes
▶  Flags: 0x012 (SYN, ACK)
   Window size value: 5840
```

Screenshot Question 7a

b) Does the lack of receiver buffer space ever throttle the sender?

The below graph shows that the receiver window grows to a maximum size close to 63,000 bytes:

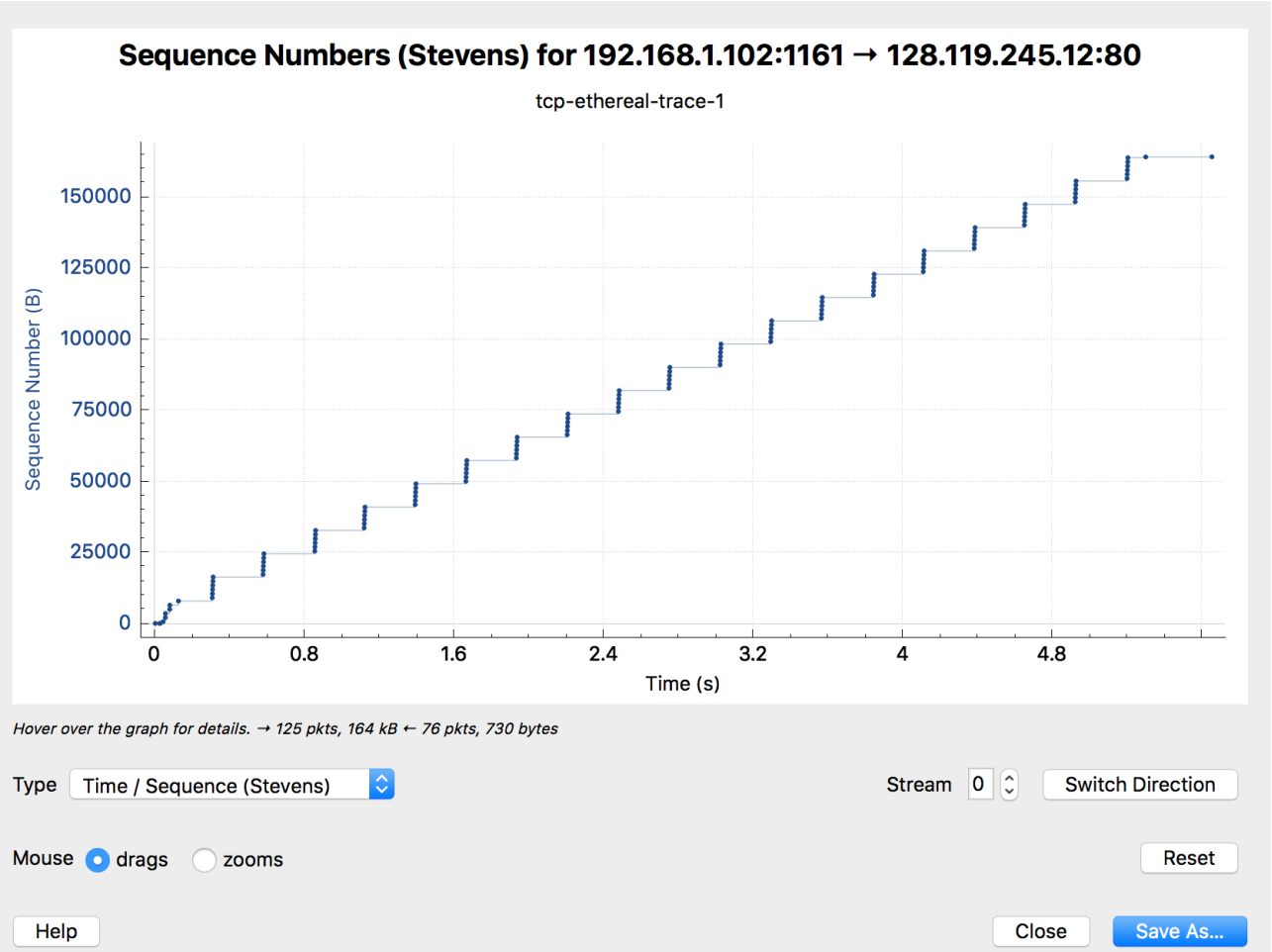


Question 7b

But nowhere near that amount of buffer space ends up being is used (seeing as packets are sent five-at-a-time before being acknowledged, and they only carry 1,460 bytes each), therefor, **throttling is not an issue.**

8. *Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?*

No, there are not. If a segment was transmitted twice it would have the same sequence number, and by checking the graph below, **the Time/Sequence (Stevens) graph**, I can see that sequence numbers increased throughout the trace.



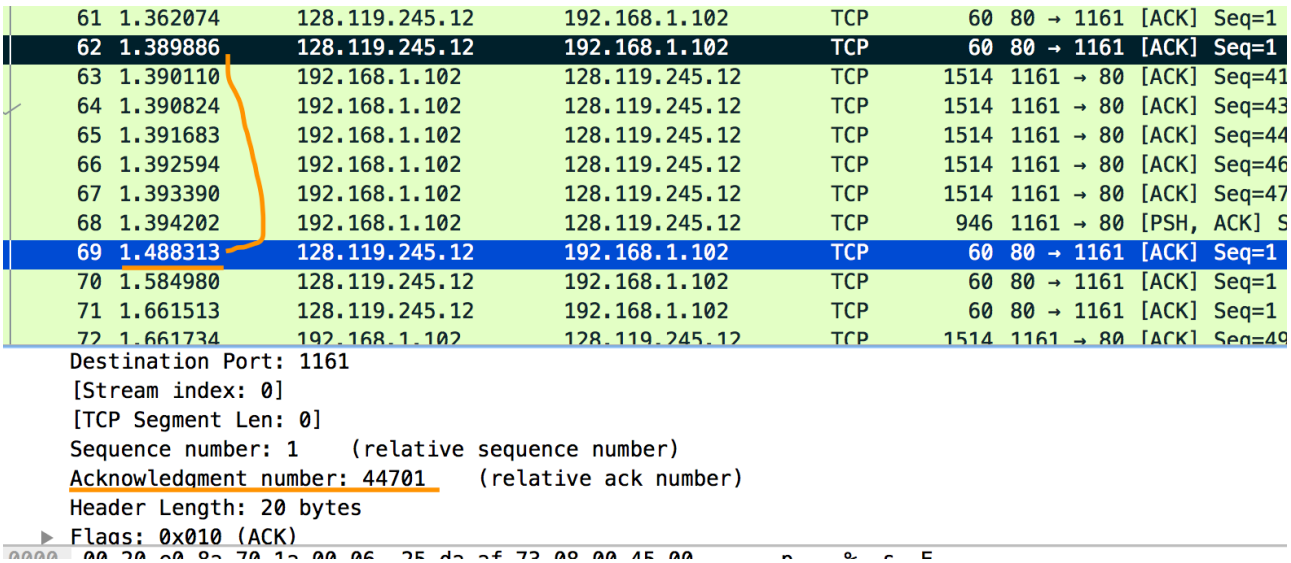
Screenshot Question 8

9. *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 247 in the text).*

The typical number of bytes acknowledged in an ack is 1460. You can see how much data was acknowledged in an ACK by subtracting its Acknowledge sequence number from the previous ACK's Aknowledge sequence number. In most cases, this number is 1460. However, in the 2 screenshot below…

62	1.389886	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=41781 Win=62780…
63	1.390110	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=41781 Ack=1 Win=17520…
64	1.390824	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=43241 Ack=1 Win=17520…
65	1.391683	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=44701 Ack=1 Win=17520…
66	1.392594	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=46161 Ack=1 Win=17520…
67	1.393390	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=47621 Ack=1 Win=17520…
68	1.394202	192.168.1.102	128.119.245.12	TCP	946 1161 → 80 [PSH, ACK] Seq=49081 Ack=1 Win=…
69	1.488313	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=44701 Win=62780…
70	1.584980	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=47621 Win=62780…
71	1.661513	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=49973 Win=62780…
72	1.661734	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=49973 Ack=1 Win=17520…

```
Destination Port: 1161
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 1    (relative sequence number)
Acknowledgment number: 41781    (relative ack number)
Header Length: 20 bytes
▸ Flags: 0x010 (ACK)
```



Looking at segments 62 and 69, which have no ACK segments between them, we see that there is a gap of *44701 – 41781* = 2920 bytes.

Segment 62 says that it is the ACK to the segment in frame 58 (TCP sequence #40889, with length 892 bytes) and segment 69 says it is the ACK to the frame in segment 64 (TCP sequence #43241 with length 1460 bytes). No ACK references the TCP segment in between 58 and 64, which is segment 63, which is sequence #41781 and has a length of 1460 bytes.

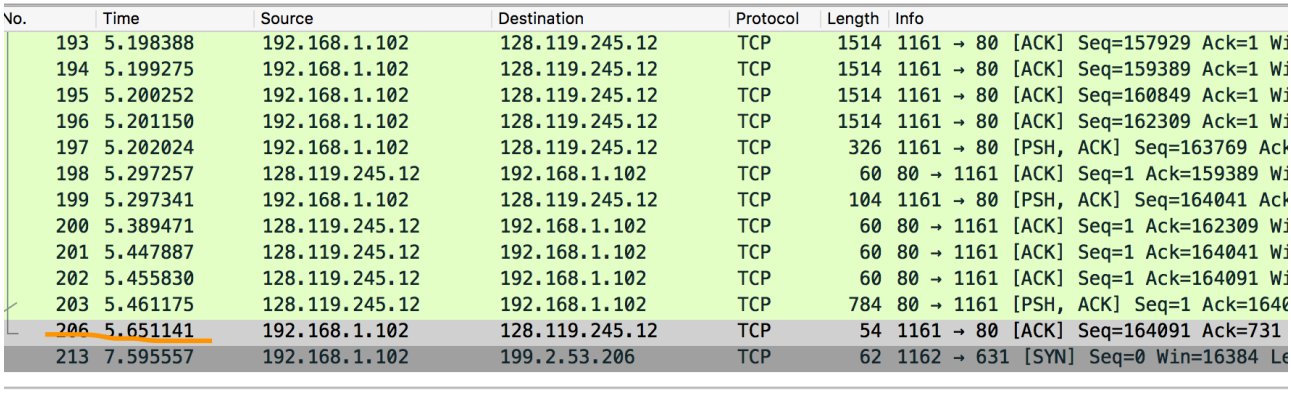
Because 1460 * 2 = 2920. It appears that the ack at segment 69 is actually acknowledging both the segments at 63 and 64.

The table in the text explains that on the event of “arrival of in-order segment with expected sequence number” the TCP receiver responds by immediately sending a single cumulative ACK, acknowledging both in-order segments.

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

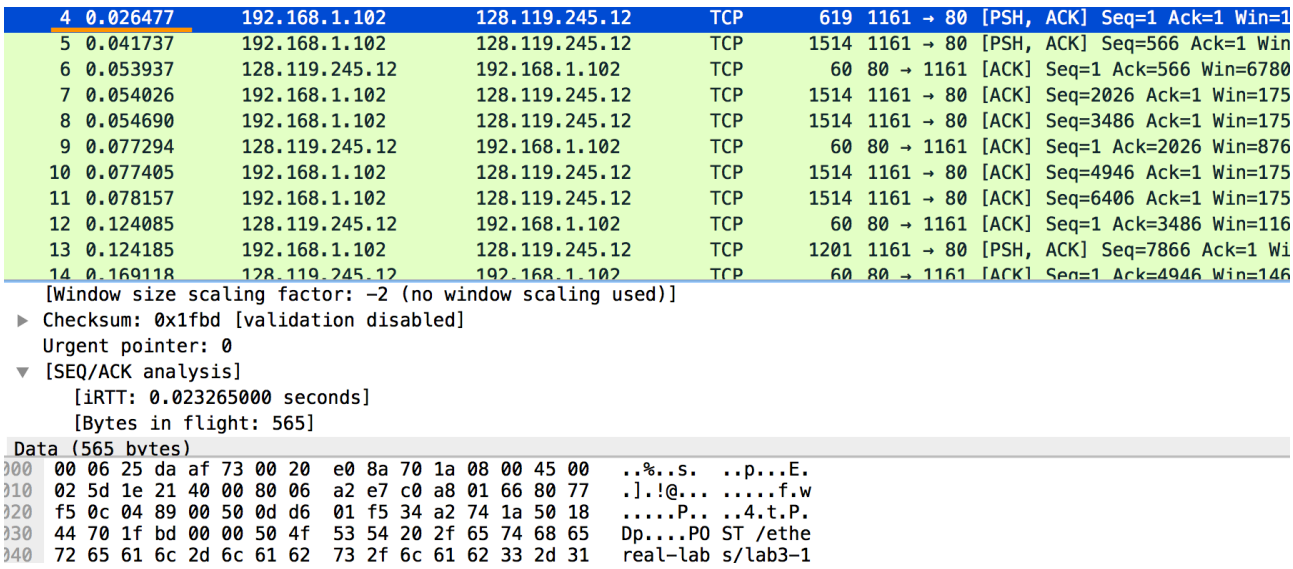
To get the throughput we divide the size of the data file being transmitted by the time it takes for the file to transmit.

The last acknowledge file arrived at 5.651141:



No.	Time	Source	Destination	Protocol	Length	Info
193	5.198388	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=157929 Ack=1 Wi
194	5.199275	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=159389 Ack=1 Wi
195	5.200252	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=160849 Ack=1 Wi
196	5.201150	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=162309 Ack=1 Wi
197	5.202024	192.168.1.102	128.119.245.12	TCP	326	1161 → 80 [PSH, ACK] Seq=163769 Ack
198	5.297257	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=159389 Wi
199	5.297341	192.168.1.102	128.119.245.12	TCP	104	1161 → 80 [PSH, ACK] Seq=164041 Ack
200	5.389471	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=162309 Wi
201	5.447887	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=164041 Wi
202	5.455830	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=164091 Wi
203	5.461175	128.119.245.12	192.168.1.102	TCP	784	80 → 1161 [PSH, ACK] Seq=1 Ack=1640
206	5.651141	192.168.1.102	128.119.245.12	TCP	54	1161 → 80 [ACK] Seq=164091 Ack=731
213	7.595557	192.168.1.102	199.2.53.206	TCP	62	1162 → 631 [SYN] Seq=0 Win=16384 Le

The transmission began with the post command at 0.026477:



No.	Time	Source	Destination	Protocol	Length	Info
4	0.026477	192.168.1.102	128.119.245.12	TCP	619	1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=1
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win
6	0.053937	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780
7	0.054026	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=2026 Ack=1 Win=175
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=3486 Ack=1 Win=175
9	0.077294	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=2026 Win=876
10	0.077405	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=4946 Ack=1 Win=175
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=6406 Ack=1 Win=175
12	0.124085	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=3486 Win=116
13	0.124185	192.168.1.102	128.119.245.12	TCP	1201	1161 → 80 [PSH, ACK] Seq=7866 Ack=1 Wi
14	0.169118	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=4946 Win=146

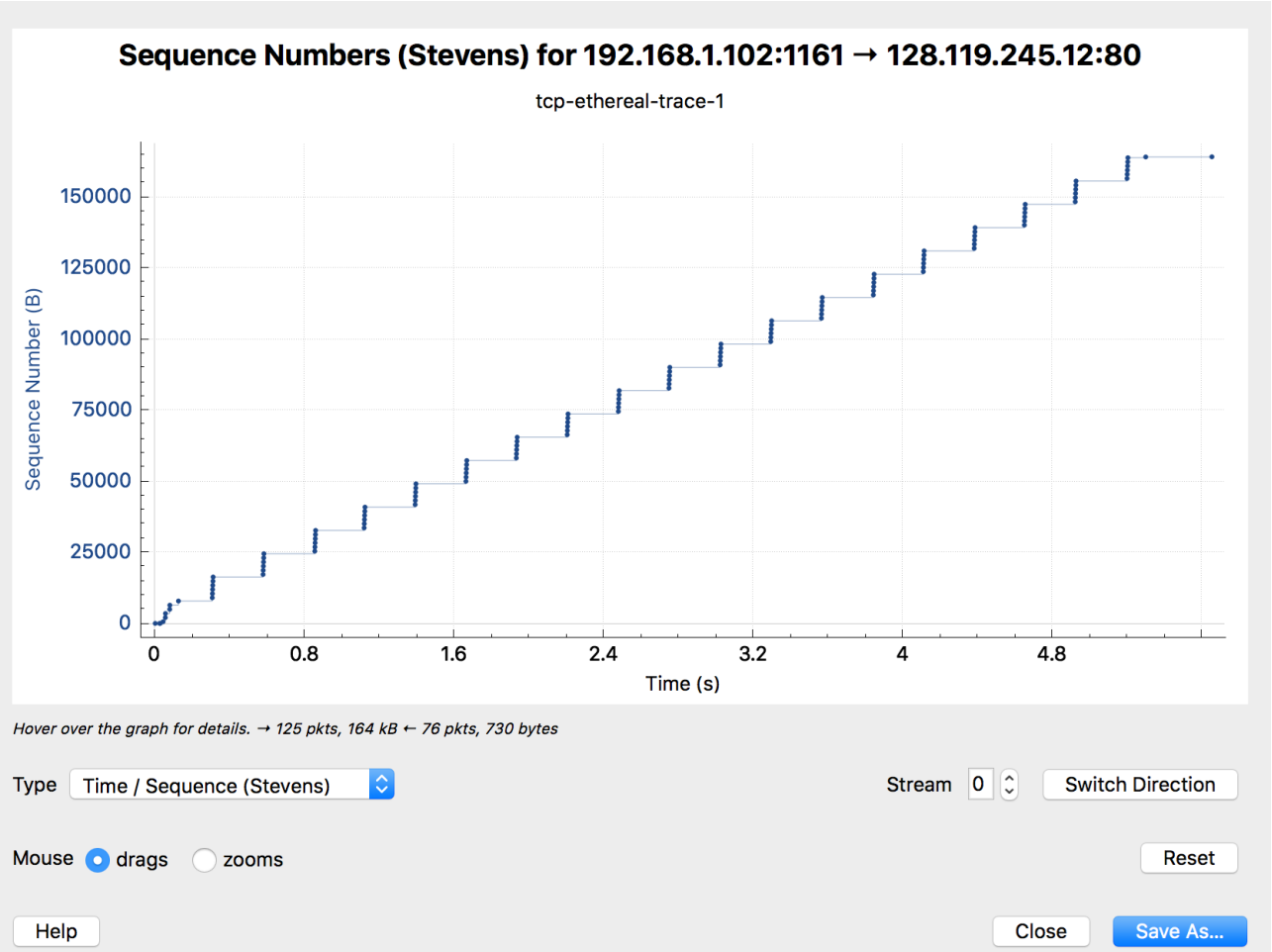
```
    [Window size scaling factor: -2 (no window scaling used)]
  ▶ Checksum: 0x1fbd [validation disabled]
    Urgent pointer: 0
  ▼ [SEQ/ACK analysis]
      [iRTT: 0.023265000 seconds]
      [Bytes in flight: 565]
Data (565 bytes)
000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s.  ..p...E.
010  02 5d 1e 21 40 00 80 06  a2 e7 c0 a8 01 66 80 77   .].!@... .....f.w
020  f5 0c 04 89 00 50 0d d6  01 f5 34 a2 74 1a 50 18   .....P.. ..4.t.P.
030  44 70 1f bd 00 00 50 4f  53 54 20 2f 65 74 68 65   Dp....PO ST /ethe
040  72 65 61 6c 2d 6c 61 62  73 2f 6c 61 62 33 2d 31   real-lab s/lab3-1
```

Therefor, the total transmission time = 5.651141 - 0.026477 = 5.624664 seconds.

The total size of the file is the final sequence number (164091) minus the sequence number when transmission began (1) = 164090 bytes.

Now, to get the kbytes/second:
164090 / 1000 = 164.09 bytes
164.09 / 5.624664 seconds ≈ **29.173298 bytes/second**.

11. *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.*



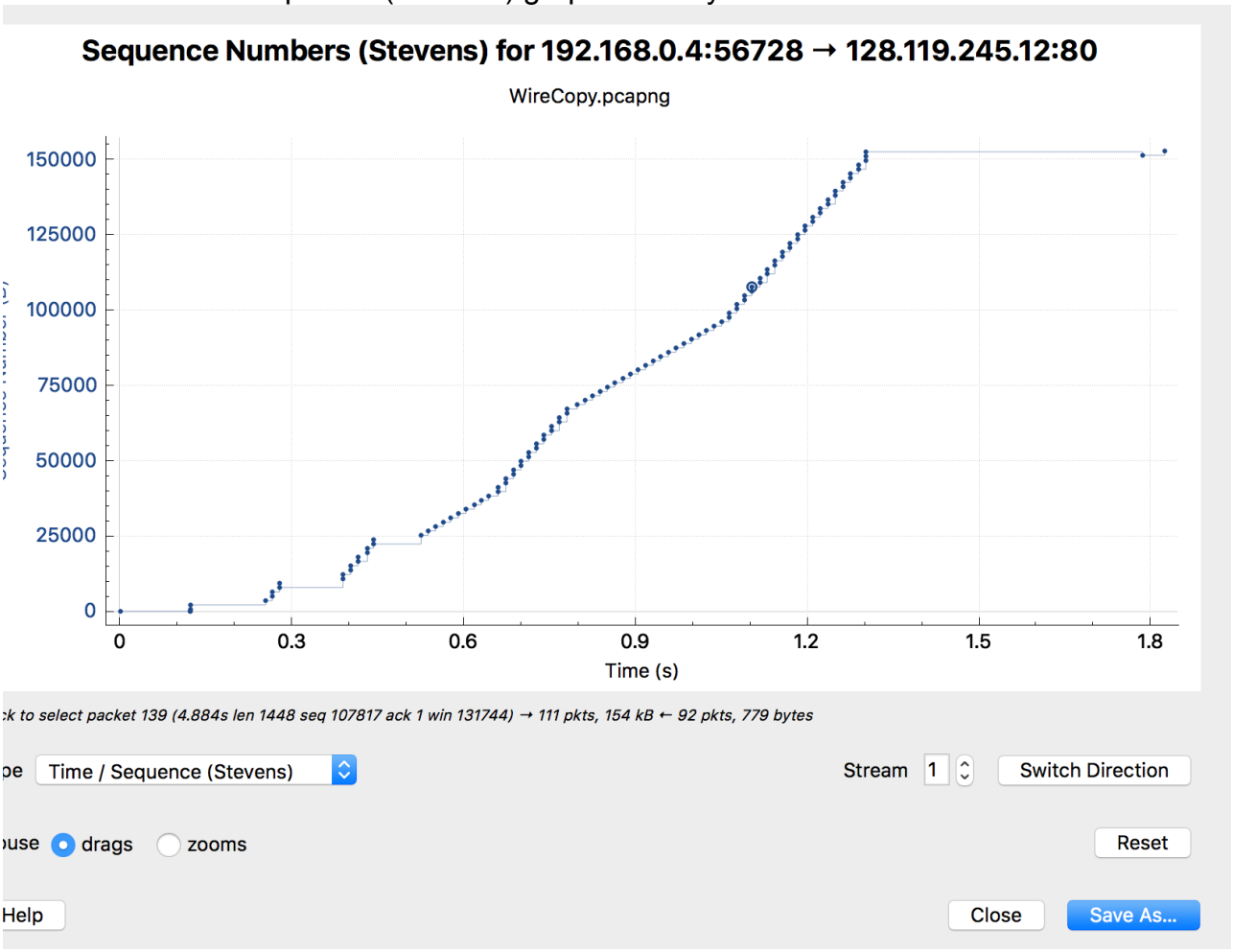
Screenshot Question 11

Looking at the graph, the last packet before the first long pause is at 0.124 seconds, so that would be my guess for how long the slowstart lasted. I cannot see exactly when the sender beings congestion control, but by inspecting the dots I can see that, after the slowstart phase, the TCP sender repeatedly sends exactly 5 packets at once, and then there is a pause. By looking at the packets and ACK files, I find that for every 5 packets that are sent, no more packets are sent until all 5 of those packets have been acknowledged.

The book describes the additive-increase, multiplicative-decrease (AIMD) method of congestion control, in which TCP linearly increases its transmission rate by increasing its congestion window size (cwnd) until segment loss is indicated by a triple-duplicate ACK event is triggered*, at which point the cwnd is halved (and then immediately begins increasing it again, probing to see if there is additional bandwidth). In contrast, in the above example, the TCP sender was ready to send data, but none was sent, perhaps because of the application that it was sent from. **3 duplicate ACKS, as described on page 248, trigger a the TCP sender to perform a fast retransmit [RFC 5681], retransmitting the missing segment before that segment's timer expires.*

12. *Answer each of two questions in **Question 11** for the trace that you have gathered when you transferred a file from your computer to gaia.cs.umass.edu*

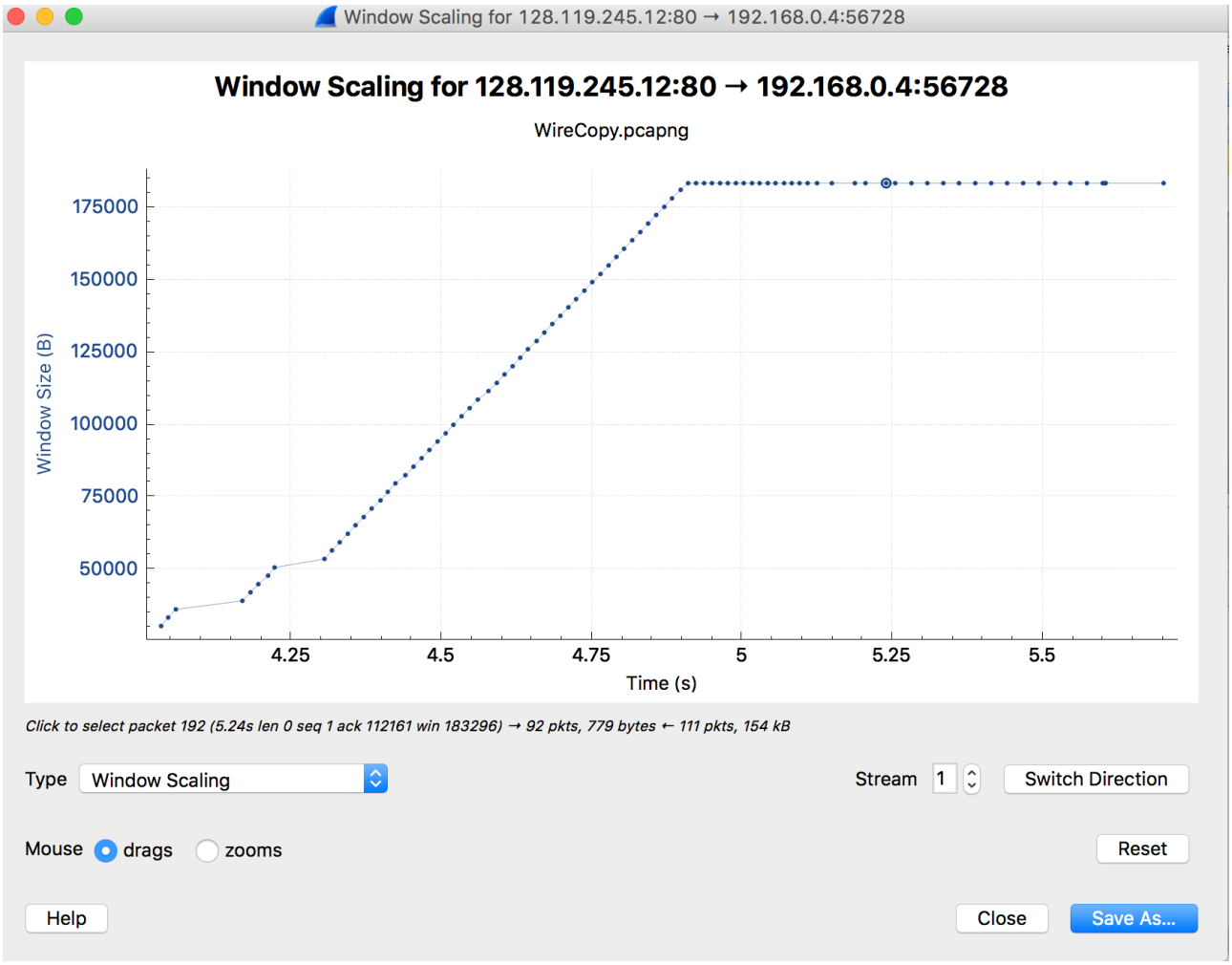
Here is the Time/Sequence (Stevens) graph from my trace:



Screenshot 1 Question 12 ***(This graph is made to to appear to start from 0 but it really starts at 3.781)***

It is harder to say how long the slowstart lasts in this case. Going by the graph above, I would have said that the slowstart appears to have lasted for about 0.45 seconds. This image also does not match the jagged-toothed AMID graph shown in the text, maybe because there was enough bandwidth to send with this time, or maybe something to do with limits imposed by the sending app.

I decided to start by taking a look at the Window Scaling graph to see the advertised buffer space from gaia.cs.umass.edu. Here is what it looks like:



Screenshot 2 Question 12

Because it closely resembles my Time/Sequence graph, I would have said that my data was being throttled by the receiver buffer space, RWND, rather than by congestion control from my end. However, that doesn't seem to be right, because there wouldn't ever end up being enough outstanding data for those numbers to be relevant, considering that the text is only 152k bytes total.

I then decided to chart my data to determine the point when the slowstart period must have ended. According the page 272 of the text, in the slow-start state, the value of cwnd begins at 1 MSS and increases by 1 MSS every time a transmitted. I found that the max segment size was 1460 bytes by looking in the SYN handshake segment:

36	4.251388	192.168.0.4	224.0.0.251	MDNS	117	Standard query 0x0000 PTR _afpovertcp._tc
37	4.251452	fe80::3e15:c2ff:fe…	ff02::fb	MDNS	137	Standard query 0x0000 PTR _afpovertcp._tc
38	4.305549	128.119.245.12	192.168.0.4	TCP	72	80 → 56728 [ACK] Seq=1409566895 Ack=37738

Then I charted my segments by looking at the sequence numbers.
Starting from the segment 10 POST:

Segment:	**Outstanding data:**
10	664 B
11	2112

12 3560

Pause to wait for segment 13, indicating that Congestion Window size here is =>3560 but < 5008…. Most likely cwdn = 1460 * 3 = 4380 B

13 (ACK 10) 2896
14 4344
15 (ACK 11) 2896
16 4344
17 5792
18 (ACK 12) 4344
19 5792
20 7240

Pause to wait for ACK segment 21, indicating that the cwnd is => 7240 but < 8688. Because we must have had a cwnd of 4380 at the time that segment 13 arrived, and there have been 3 ACKs since, assuming we are still in slow-start mode, it must now be (3 * 1460) + 4380 = 8670.

21 (ACK 13 AND 14) 4344
22 5792
23 7240
24(ACK 15 AND 16) 4344
25 5792
26 7240
27(ACK 17) 5792
28 7240
29 8688
30 (ACK 18/19) 5792
31 7240
32 8688
33 (ACK 20) 7240
34 8688
35 10136

This is where the last long pause is, while we wait for ACK 38. We can determine that the cwnd must be => 10136, but less than 11584. Since we last calculated the cwnd to be 8670, there have been 8 more packages acknowledges across 5 more ACKs. Being conservative and only adding one MSS for each segment we've received, (1460 * 5) + 8670 = 15970, which must be higher than our cwnd. Therefor, at this point, we are no longer in the slow-start period, and congestion control has begun.