CSE Lab 4

WireShark TCP Lab

Aditya Balwani, SBUID : 109353920

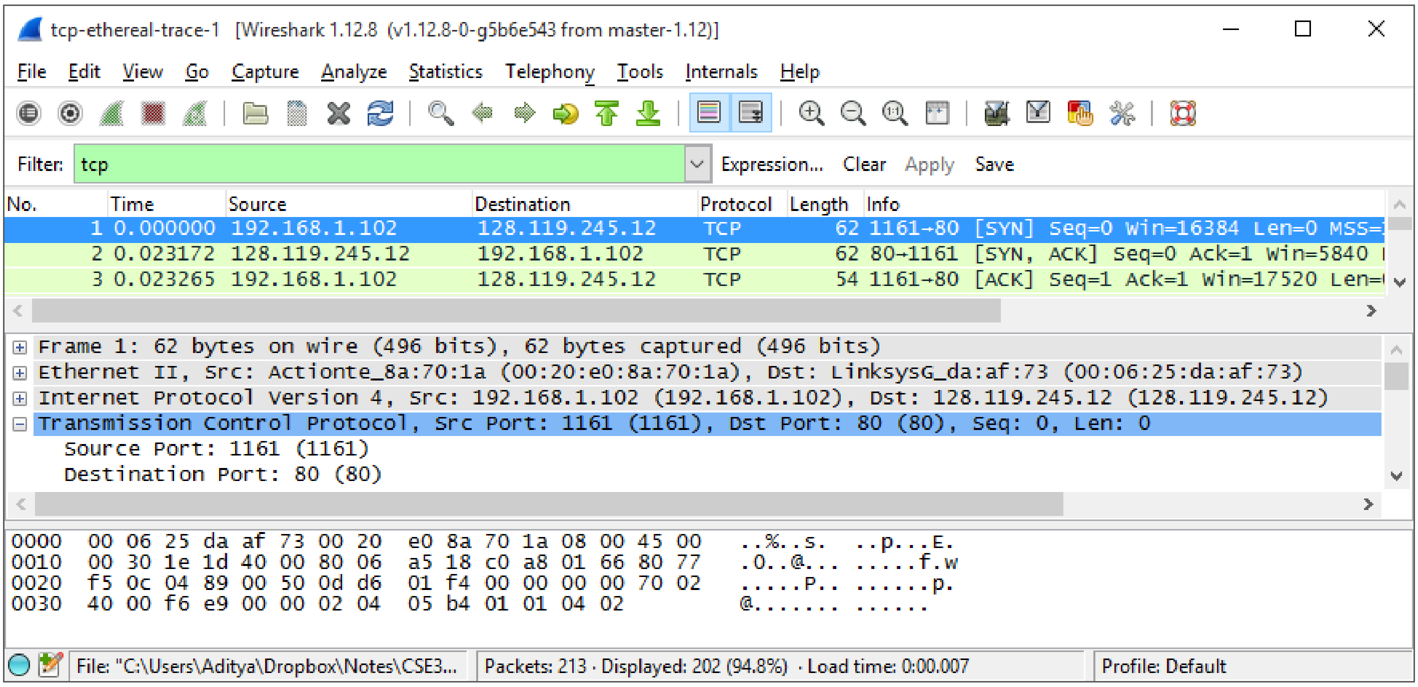
Part 1

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?**

The client computer is 192.168.1.102 and uses the TCP port 1161

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?**

The IP address of gaia.cs.umass.edu is 128.119.249.12 and is sending and receiving TCP segments on port 80.


tcp-ethereal-trace-1 [Wireshark 1.12.8 (v1.12.8-0-g5b6e543 from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: tcp

```
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=
```

```
⊞ Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 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 (192.168.1.102), Dst: 128.119.245.12 (128.119.245.12)
⊟ Transmission Control Protocol, Src Port: 1161 (1161), Dst Port: 80 (80), Seq: 0, Len: 0
    Source Port: 1161 (1161)
    Destination Port: 80 (80)
```

```
0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s.  ..p...E.
0010  00 30 1e 1d 40 00 80 06  a5 18 c0 a8 01 66 80 77   .0..@...  .....f.w
0020  f5 0c 04 89 00 50 0d d6  01 f4 00 00 00 00 70 02   .....P..  ......p.
0030  40 00 f6 e9 00 00 02 04  05 b4 01 01 04 02         @.......  ......
```

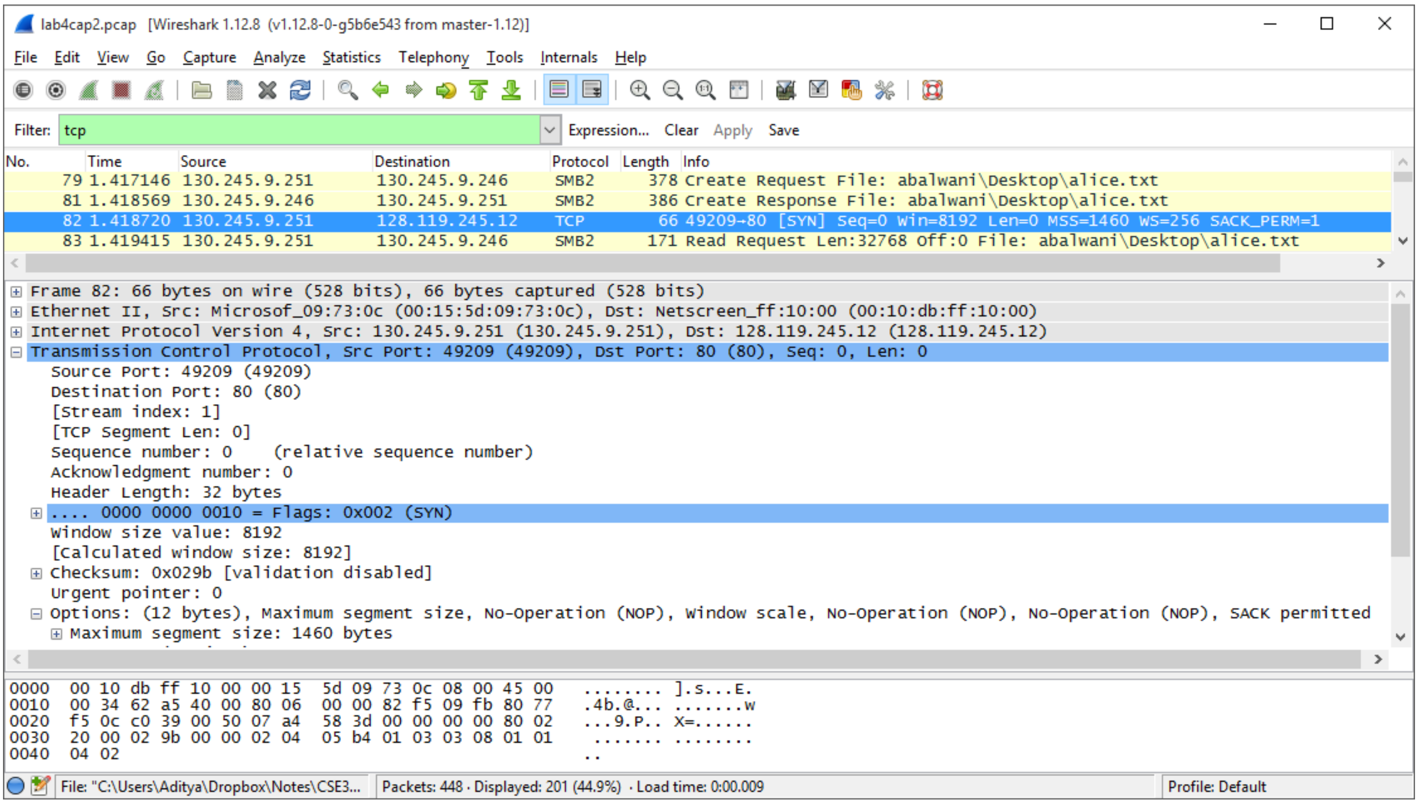
File: "C:\Users\Aditya\Dropbox\Notes\CSE3... Packets: 213 · Displayed: 202 (94.8%) · Load time: 0:00.007 Profile: Default

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

The client computer is 130.245.9.251 and is using port number 49209

4. **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 of the TCP SYN segment is 0. We can tell its a SYN segment by looking at its flag where the second bit represents Syn and it is set.


lab4cap2.pcap [Wireshark 1.12.8 (v1.12.8-0-g5b6e543 from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: tcp

```
No.   Time     Source         Destination     Protocol Length Info
   79 1.417146 130.245.9.251  130.245.9.246   SMB2       378 Create Request File: abalwani\Desktop\alice.txt
   81 1.418569 130.245.9.246  130.245.9.251   SMB2       386 Create Response File: abalwani\Desktop\alice.txt
   82 1.418720 130.245.9.251  128.119.245.12  TCP         66 49209→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
   83 1.419415 130.245.9.251  130.245.9.246   SMB2       171 Read Request Len:32768 Off:0 File: abalwani\Desktop\alice.txt
```

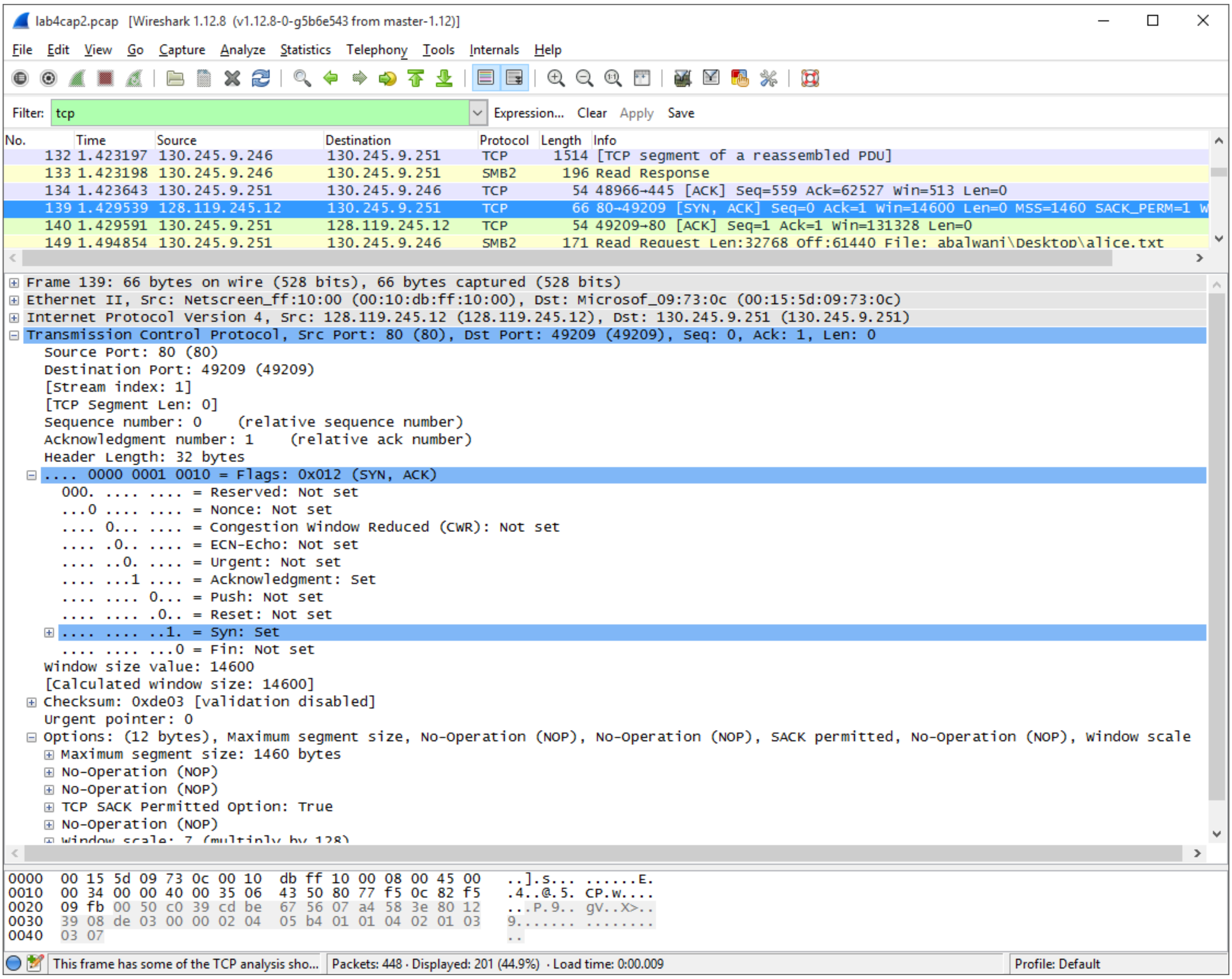
```
⊞ Frame 82: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: Microsof_09:73:0c (00:15:5d:09:73:0c), Dst: Netscreen_ff:10:00 (00:10:db:ff:10:00)
⊞ Internet Protocol Version 4, Src: 130.245.9.251 (130.245.9.251), Dst: 128.119.245.12 (128.119.245.12)
⊟ Transmission Control Protocol, Src Port: 49209 (49209), Dst Port: 80 (80), Seq: 0, Len: 0
    Source Port: 49209 (49209)
    Destination Port: 80 (80)
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 0
    Header Length: 32 bytes
  ⊞ .... 0000 0000 0010 = Flags: 0x002 (SYN)
    Window size value: 8192
    [Calculated window size: 8192]
  ⊞ Checksum: 0x029b [validation disabled]
    Urgent pointer: 0
  ⊟ Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
    ⊞ Maximum segment size: 1460 bytes
```

```
0000  00 10 db ff 10 00 00 15  5d 09 73 0c 08 00 45 00   ........ ].s...E.
0010  00 34 62 a5 40 00 80 06  00 00 82 f5 09 fb 80 77   .4b.@... .......w
0020  f5 0c c0 39 00 50 07 a4  58 3d 00 00 00 00 80 02   ...9.P.. X=......
0030  20 00 02 9b 00 00 02 04  05 b4 01 03 03 08 01 01    ....... ........
0040  04 02                                              ..
```

File: "C:\Users\Aditya\Dropbox\Notes\CSE3... Packets: 448 · Displayed: 201 (44.9%) · Load time: 0:00.009 Profile: Default

5. **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 of the TCP SYNACK segment is 0. The value of the Acknowledgement Number is 1. The value is determined by adding one to the SYN segment that the client sent to gaia. It is identified as a SYNACK as both the the SYN and the ACK bits in the flags are set.


lab4cap2.pcap [Wireshark 1.12.8 (v1.12.8-0-g5b6e543 from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply Save

```
No.     Time     Source          Destination     Protocol Length Info
    132 1.423197 130.245.9.246   130.245.9.251   TCP       1514 [TCP segment of a reassembled PDU]
    133 1.423198 130.245.9.246   130.245.9.251   SMB2       196 Read Response
    134 1.423643 130.245.9.251   130.245.9.246   TCP         54 48966→445 [ACK] Seq=559 Ack=62527 Win=513 Len=0
    139 1.429539 128.119.245.12  130.245.9.251   TCP         66 80→49209 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 W
    140 1.429591 130.245.9.251   128.119.245.12  TCP         54 49209→80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
    149 1.494854 130.245.9.251   130.245.9.246   SMB2       171 Read Request Len:32768 Off:61440 File: abalwani\Desktop\alice.txt
```

```
⊞ Frame 139: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: Netscreen_ff:10:00 (00:10:db:ff:10:00), Dst: Microsof_09:73:0c (00:15:5d:09:73:0c)
⊞ Internet Protocol Version 4, Src: 128.119.245.12 (128.119.245.12), Dst: 130.245.9.251 (130.245.9.251)
⊟ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 49209 (49209), Seq: 0, Ack: 1, Len: 0
    Source Port: 80 (80)
    Destination Port: 49209 (49209)
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header Length: 32 bytes
  ⊟ .... 0000 0001 0010 = Flags: 0x012 (SYN, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    ⊞ .... .... ..1. = Syn: Set
      .... .... ...0 = Fin: Not set
    Window size value: 14600
    [Calculated window size: 14600]
  ⊞ Checksum: 0xde03 [validation disabled]
    Urgent pointer: 0
  ⊟ Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale
    ⊞ Maximum segment size: 1460 bytes
    ⊞ No-Operation (NOP)
    ⊞ No-Operation (NOP)
    ⊞ TCP SACK Permitted Option: True
    ⊞ No-Operation (NOP)
    ⊞ Window scale: 7 (multiply by 128)
```

```
0000  00 15 5d 09 73 0c 00 10  db ff 10 00 08 00 45 00   ..].s... ......E.
0010  00 34 00 00 40 00 35 06  43 50 80 77 f5 0c 82 f5   .4..@.5. CP.w....
0020  09 fb 00 50 c0 39 cd be  67 56 07 a4 58 3e 80 12   ...P.9.. gV..X>..
0030  39 08 de 03 00 00 02 04  05 b4 01 01 04 02 01 03   9....... ........
0040  03 07                                              ..
```

This frame has some of the TCP analysis sho... | Packets: 448 · Displayed: 201 (44.9%) · Load time: 0:00.009 | Profile: Default

6. **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 of the TCP segment containing the HTTP POST command is 145190.

7. **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.**

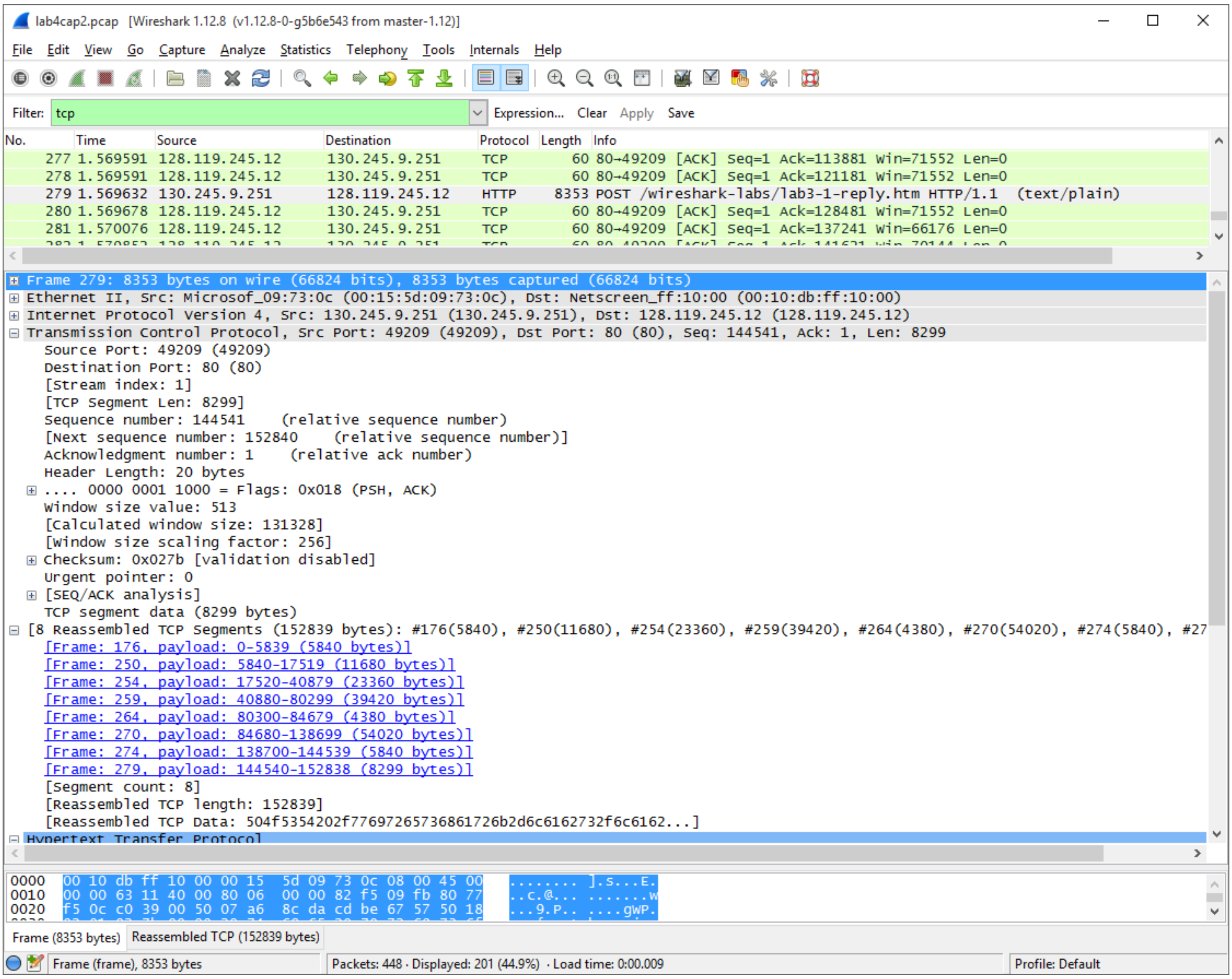
The HTTP Post is considered the first segment. Segments 1 - 6 are 176, 250, 254, 259, 264 and 270 and the corresponding ACKS are 6, 9, 14, 15, 16, 17

Estimated RTT = 0.875 * EstimatedRTT + 0.125 * SampleRTT

Num	Seq. Number	Length (bytes)	Time Sent	Time Ack Recieved	RTT (s)	EstimatedRTT (s)
1	1	5840	1.509094	1.52552	0.016426	0.016426
2	5841	11680	1.52556	1.536058	0.010498	0.016426
3	17521	23360	1.536111	1.547570	0.011459	0.015685
4	40881	39474	1.547656	1.558070	0.010414	0.01515675
5	80301	4434	1.558109	1.558569	0.00046	0.014563
6	84681	54020	1.558634	1.568623	0.009989	0.0128001

8. **What is the length of each of the first six TCP segments?**

Segment 1 : 5840 bytes
Segment 2 : 11680 bytes
Segment 3 : 23360 bytes
Segment 4 : 39474 bytes
Segment 5 : 4434 bytes
Segment 6 : 54020 bytes


lab4cap2.pcap [Wireshark 1.12.8 (v1.12.8-0-g5b6e543 from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply Save

```
No.     Time     Source          Destination     Protocol Length Info
    277 1.569591 128.119.245.12  130.245.9.251   TCP        60 80→49209 [ACK] Seq=1 Ack=113881 Win=71552 Len=0
    278 1.569591 128.119.245.12  130.245.9.251   TCP        60 80→49209 [ACK] Seq=1 Ack=121181 Win=71552 Len=0
    279 1.569632 130.245.9.251   128.119.245.12  HTTP     8353 POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1  (text/plain)
    280 1.569678 128.119.245.12  130.245.9.251   TCP        60 80→49209 [ACK] Seq=1 Ack=128481 Win=71552 Len=0
    281 1.570076 128.119.245.12  130.245.9.251   TCP        60 80→49209 [ACK] Seq=1 Ack=137241 Win=66176 Len=0
```

```
⊞ Frame 279: 8353 bytes on wire (66824 bits), 8353 bytes captured (66824 bits)
⊞ Ethernet II, Src: Microsof_09:73:0c (00:15:5d:09:73:0c), Dst: Netscreen_ff:10:00 (00:10:db:ff:10:00)
⊞ Internet Protocol Version 4, Src: 130.245.9.251 (130.245.9.251), Dst: 128.119.245.12 (128.119.245.12)
⊟ Transmission Control Protocol, Src Port: 49209 (49209), Dst Port: 80 (80), Seq: 144541, Ack: 1, Len: 8299
    Source Port: 49209 (49209)
    Destination Port: 80 (80)
    [Stream index: 1]
    [TCP Segment Len: 8299]
    Sequence number: 144541    (relative sequence number)
    [Next sequence number: 152840    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header Length: 20 bytes
  ⊞ .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
    Window size value: 513
    [Calculated window size: 131328]
    [Window size scaling factor: 256]
  ⊞ Checksum: 0x027b [validation disabled]
    Urgent pointer: 0
  ⊞ [SEQ/ACK analysis]
    TCP segment data (8299 bytes)
⊟ [8 Reassembled TCP Segments (152839 bytes): #176(5840), #250(11680), #254(23360), #259(39420), #264(4380), #270(54020), #274(5840), #27
    [Frame: 176, payload: 0-5839 (5840 bytes)]
    [Frame: 250, payload: 5840-17519 (11680 bytes)]
    [Frame: 254, payload: 17520-40879 (23360 bytes)]
    [Frame: 259, payload: 40880-80299 (39420 bytes)]
    [Frame: 264, payload: 80300-84679 (4380 bytes)]
    [Frame: 270, payload: 84680-138699 (54020 bytes)]
    [Frame: 274, payload: 138700-144539 (5840 bytes)]
    [Frame: 279, payload: 144540-152838 (8299 bytes)]
    [Segment count: 8]
    [Reassembled TCP length: 152839]
    [Reassembled TCP Data: 504f5354202f77697265736861726b2d6c6162732f6c6162...]
⊟ Hypertext Transfer Protocol
```

```
0000  00 10 db ff 10 00 00 15  5d 09 73 0c 08 00 45 00   ........ ].s...E.
0010  00 00 63 11 40 00 80 06  00 00 82 f5 09 fb 80 77   ..c.@... .......w
0020  f5 0c c0 39 00 50 07 a6  8c da cd be 67 57 50 18   ...9.P.. ....gWP.
```

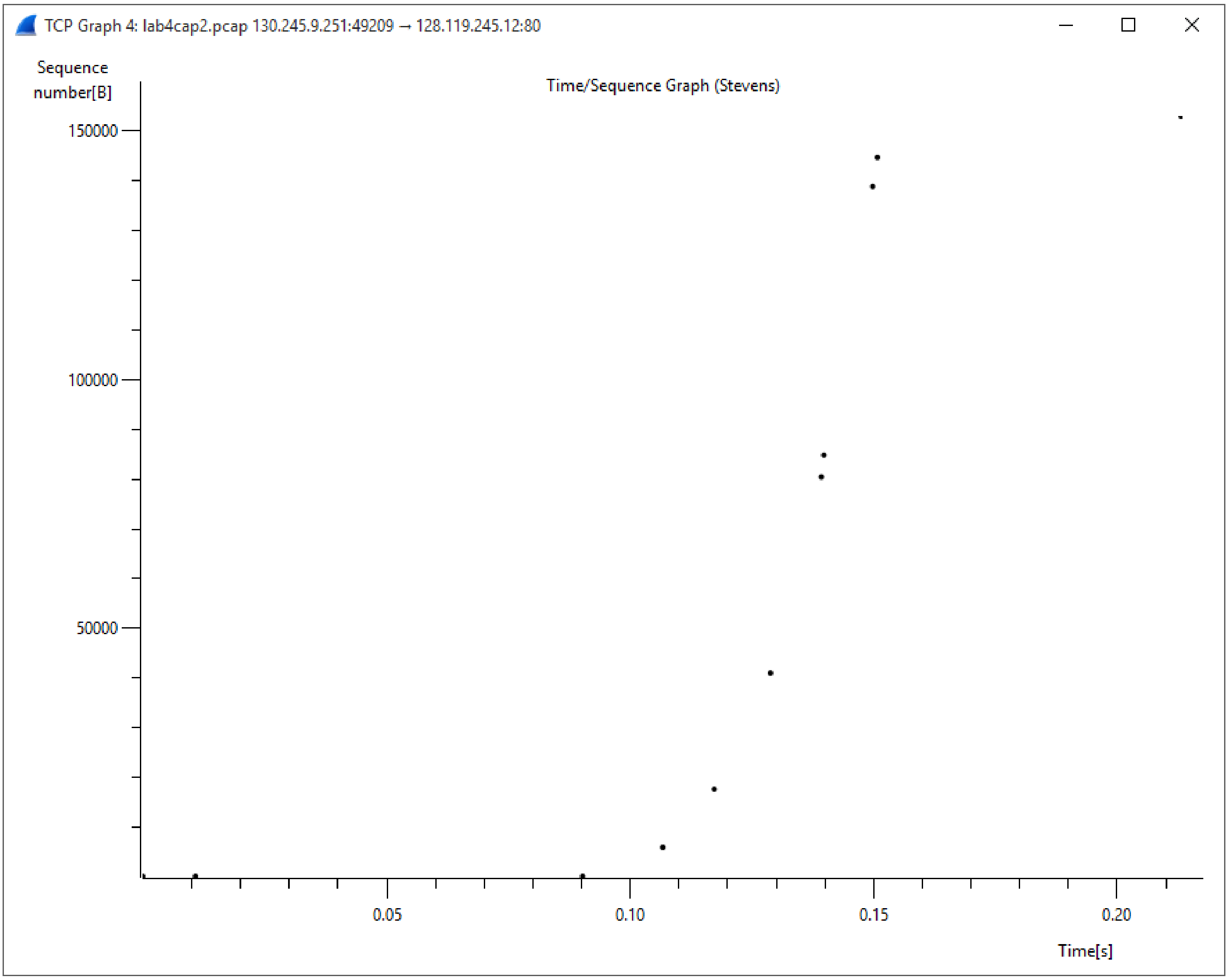
Frame (8353 bytes)
Reassembled TCP (152839 bytes)
Frame (frame), 8353 bytes
Packets: 448 · Displayed: 201 (44.9%) · Load time: 0:00.009
Profile: Default

9. **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?**

The minimum amount of available buffer space advertised at the beginning is 14600 bytes. The lack of reciever buffer space never throttles the user because it never reaches the maximum

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

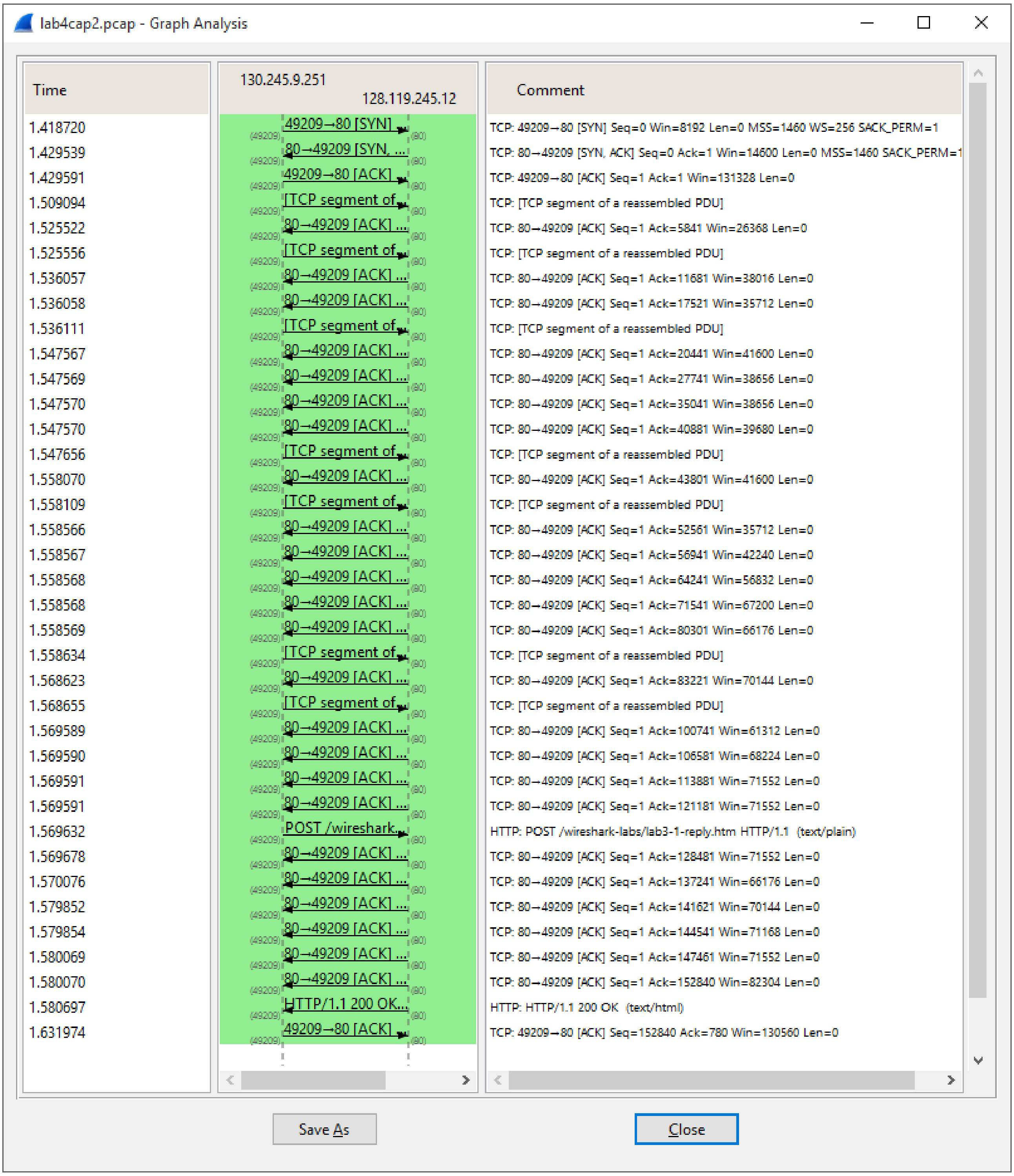
No there are no retransmitted segments in this trace file. To check this we check the TCP Time Graph (Stevens). As we can see from the graph, the sequence numbers steadily increase and no sequence number is repeated so no segment is repeated


TCP Graph 4: lab4cap2.pcap 130.245.9.251:49209 → 128.119.245.12:80
Time/Sequence Graph (Stevens)
Sequence number[B]
150000
100000
50000
0.05
0.10
0.15
0.20
Time[s]

11. **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 receiver typically acknowledges 1460 bytes in an ACK.

In our data we never see the situation where every other segment is acked however that is because our data segments are displayed as a few big segnments in Wireshark when in reality is a bunch of smaller segments, so we see a lot more acks.



Time	130.245.9.251 128.119.245.12	Comment
1.418720	49209→80 [SYN]	TCP: 49209→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
1.429539	80→49209 [SYN, ...	TCP: 80→49209 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1
1.429591	49209→80 [ACK]	TCP: 49209→80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
1.509094	[TCP segment of...	TCP: [TCP segment of a reassembled PDU]
1.525522	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=5841 Win=26368 Len=0
1.525556	[TCP segment of...	TCP: [TCP segment of a reassembled PDU]
1.536057	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=11681 Win=38016 Len=0
1.536058	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=17521 Win=35712 Len=0
1.536111	[TCP segment of...	TCP: [TCP segment of a reassembled PDU]
1.547567	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=20441 Win=41600 Len=0
1.547569	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=27741 Win=38656 Len=0
1.547570	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=35041 Win=38656 Len=0
1.547570	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=40881 Win=39680 Len=0
1.547656	[TCP segment of...	TCP: [TCP segment of a reassembled PDU]
1.558070	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=43801 Win=41600 Len=0
1.558109	[TCP segment of...	TCP: [TCP segment of a reassembled PDU]
1.558566	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=52561 Win=35712 Len=0
1.558567	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=56941 Win=42240 Len=0
1.558568	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=64241 Win=56832 Len=0
1.558568	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=71541 Win=67200 Len=0
1.558569	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=80301 Win=66176 Len=0
1.558634	[TCP segment of...	TCP: [TCP segment of a reassembled PDU]
1.568623	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=83221 Win=70144 Len=0
1.568655	[TCP segment of...	TCP: [TCP segment of a reassembled PDU]
1.569589	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=100741 Win=61312 Len=0
1.569590	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=106581 Win=68224 Len=0
1.569591	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=113881 Win=71552 Len=0
1.569591	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=121181 Win=71552 Len=0
1.569632	POST /wireshark...	HTTP: POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1 (text/plain)
1.569678	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=128481 Win=71552 Len=0
1.570076	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=137241 Win=66176 Len=0
1.579852	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=141621 Win=70144 Len=0
1.579854	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=144541 Win=71168 Len=0
1.580069	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=147461 Win=71552 Len=0
1.580070	80→49209 [ACK] ...	TCP: 80→49209 [ACK] Seq=1 Ack=152840 Win=82304 Len=0
1.580697	HTTP/1.1 200 OK...	HTTP: HTTP/1.1 200 OK (text/html)
1.631974	49209→80 [ACK] ...	TCP: 49209→80 [ACK] Seq=152840 Ack=780 Win=130560 Len=0

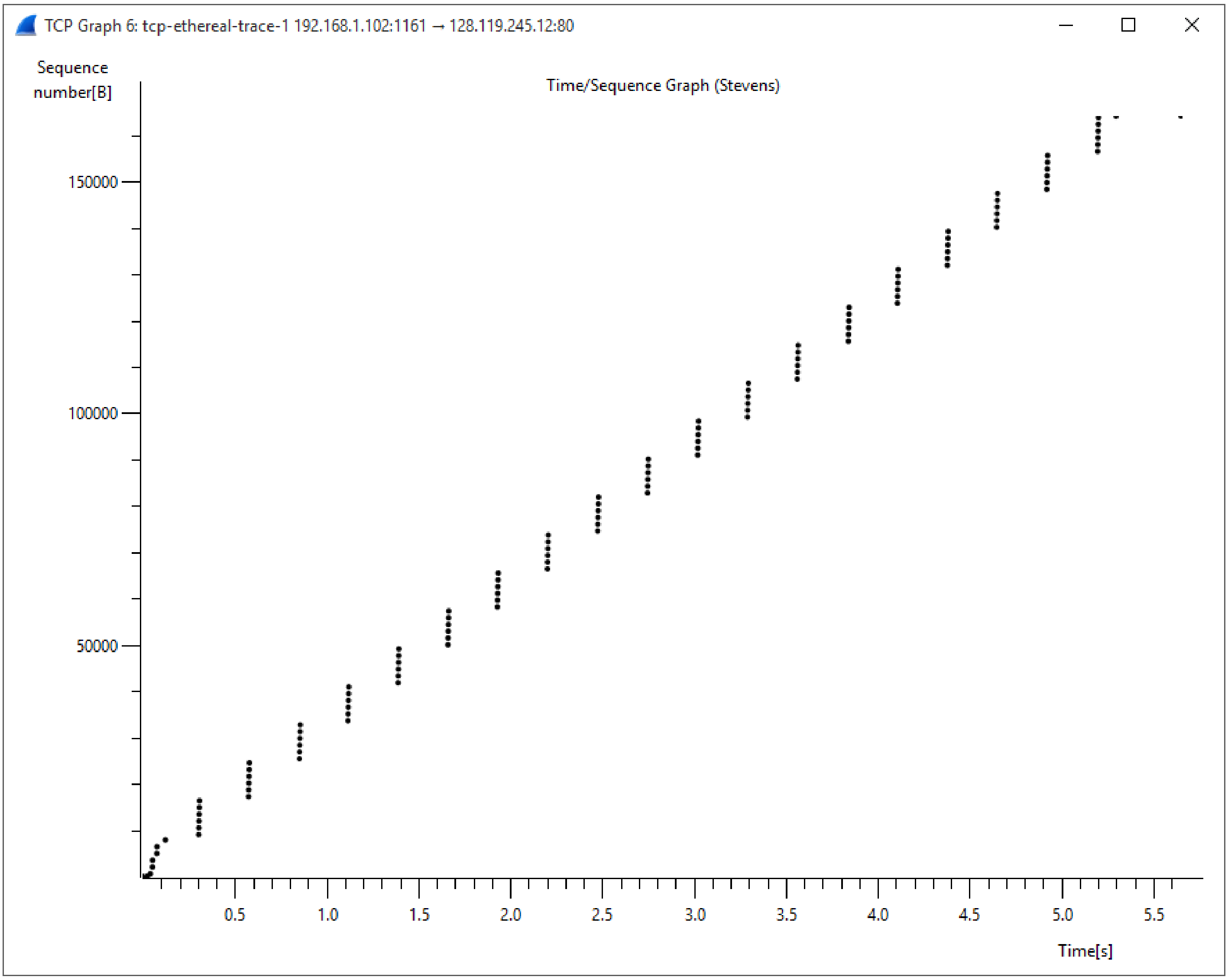
12. **What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.**

To calculate the throughput we need the total number of bytes transferred which we can get by looking at the difference between the sequence numbers between the first and last segments and then we divide that with the total time which is the time difference between the first and last segment.
The total size is : 152840 - 1 = 152839 bytes Total time is : 1.580070 - 1.509094 = 0.070976s

Throughput is then 2.05 MBps

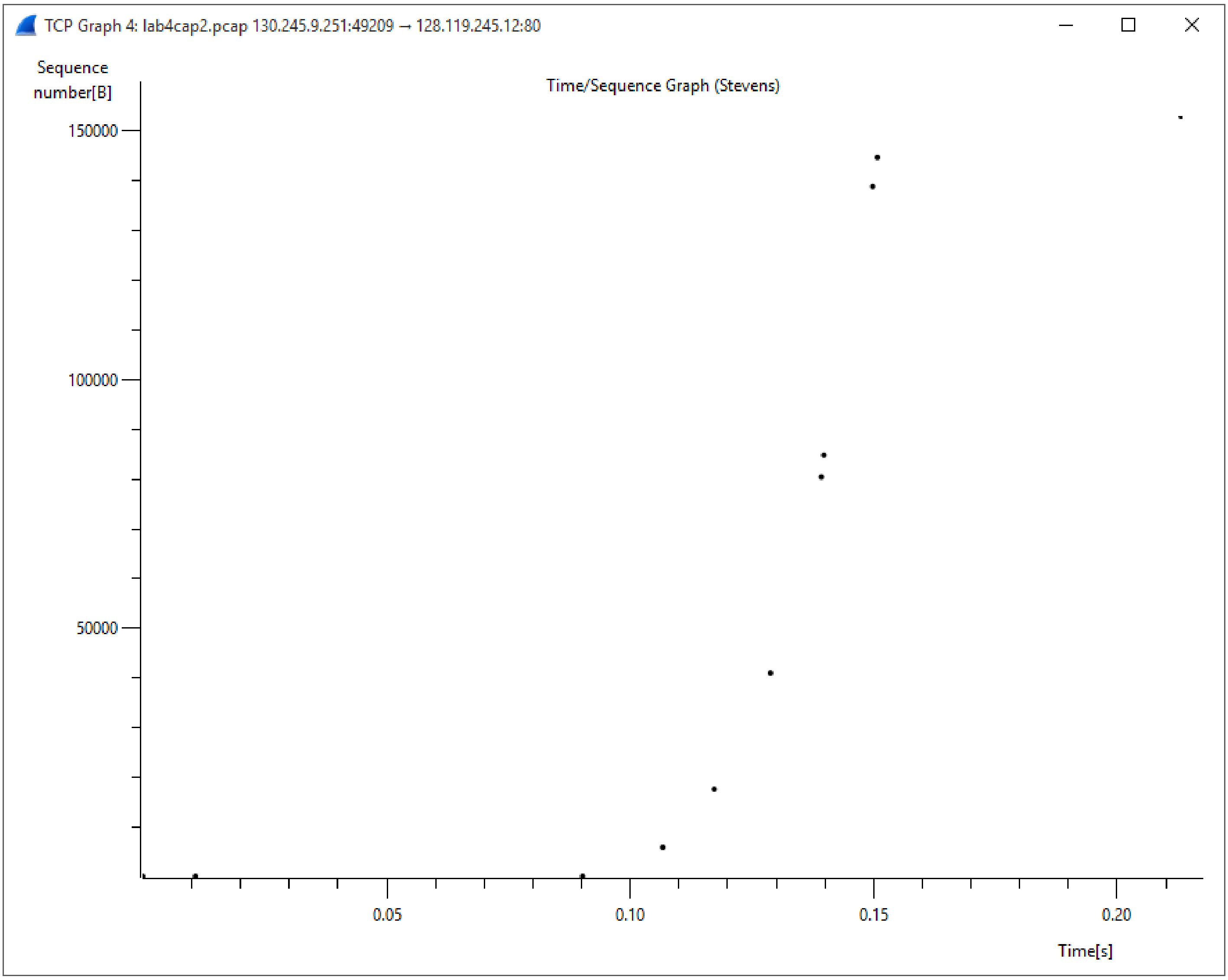
13. **Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.**

The slowstart phase only lasts for the first 0.1 seconds and after that the congestion control takes over. However its not the linear graph that we expect from ideal data. Instead the segments seem to be transmitted in groups of 5.


TCP Graph 6: tcp-ethereal-trace-1 192.168.1.102:1161 → 128.119.245.12:80
Time/Sequence Graph (Stevens)
Sequence number[B]
150000
100000
50000
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
Time[s]

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

In case of our data, since the internet is a lot faster, the whole file gets transmitted before the connections leaves the slow-start phase. As a result the whole graph represents the slow-start which again lasts for 0.15s


TCP Graph 4: lab4cap2.pcap 130.245.9.251:49209 → 128.119.245.12:80
Time/Sequence Graph (Stevens)
Sequence number[B]
150000
100000
50000
0.05
0.10
0.15
0.20
Time[s]

Part 2

Documentation

This is a simple ping client written to simulate packet loss that takes place over UDP communications. To run this first start the server by running `python Lab4PingServer.py` and then running the client on another machine with the server host address and port number as arguments for example `python Lab4PingClient.py localhost 8920`
http://allv24.all.cs.stonybrook.edu:8920/Hello.html

Server Code

```
# pylint: disable=W,C

# UDP Server
import random
from socket import *

serverSocket = socket(AF_INET, SOCK_DGRAM)

serverSocket.bind(('',8920))

while True:
    rand = random.randint(0,10)

    message, address = serverSocket.recvfrom(1024)

    message.upper()

    if rand < 4:
        continue

    serverSocket.sendto(message, address)
```

Client Code

```
#pylint: disable = W,C

# Aditya Balwani
# SBUID : 109353920
# CSE 310 Lab 4 Python Client

import sys, time
from socket import *
from datetime import datetime

# Get the server hostname and port as command line arguments
```

```
argv = sys.argv
host = argv[1]
port = argv[2]

# Create UDP client socket
clientSocket = socket(AF_INET, SOCK_DGRAM)
# Set socket timeout as 1 second
clientSocket.settimeout(2)

# Command line argument is a string, change the port into integer
port = int(port)

print "Pinging " + host+":"+str(port) + " with 10 packets"

# Init initial values
pingMin = 1
pingMax = 0
pingSum = 0
successCount = 0

for seqno in range(0,10):
# Fill in end

    # Format the message to be sent
    data = "Ping " + str(seqno) + " " + time.asctime()

    # Attempt to send and receive
    try:
     # Record the sent time
        sntTime = time.time()

     # Send the UDP packet with the ping message
        clientSocket.sendto(data,(host,port))

     # Receive the server response
        modifiedSentence,sender = clientSocket.recvfrom(1024)
        clientSocket.settimeout(2)

        # Record the received time
        rcvTime = time.time()

        #Do required calculations
        responseTime = rcvTime - sntTime

        if pingMin > responseTime:
            pingMin = responseTime
        if pingMax < responseTime:
            pingMax = responseTime
        successCount += 1
        pingSum += responseTime


     # Display the server address and server response as an output
        print "\nReply from " + str(sender) + "\nMessage : " + modifiedSentence

     # Round trip time is the difference between sent and received time
        print "RTT: " + str(rcvTime - sntTime) + " seconds"

    except Exception,err:
        # print(err)
        # Server does not response
 # Assume the packet is lost
        print "\nRequest timed out."

# Close the client socket
clientSocket.close()

# Print statistics
print "\nMinimum Response Time : " + str(pingMin) + " seconds"
print "Maximum Response Time : " + str(pingMax) + " seconds"
print "Average Response Time : " + str(pingSum/successCount) + " seconds"
print "Success Rate : " + str(successCount*10) + "%"
```