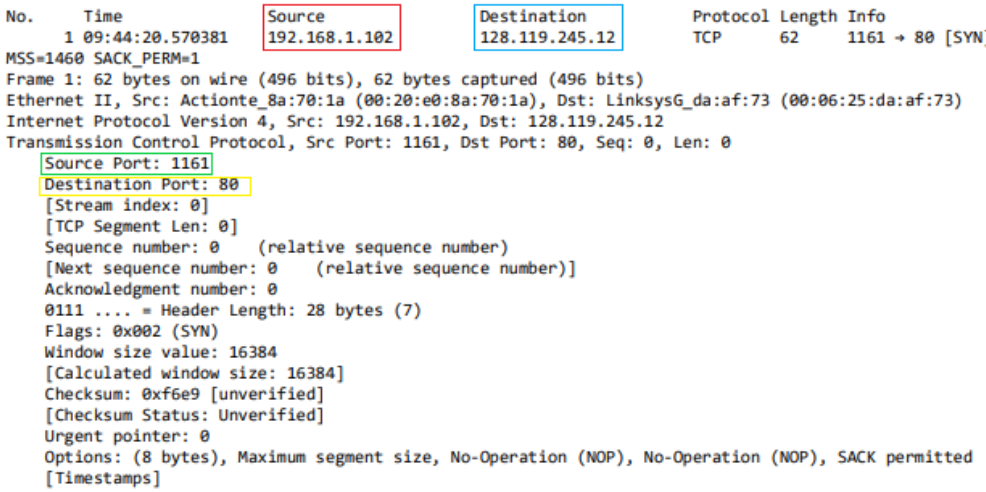
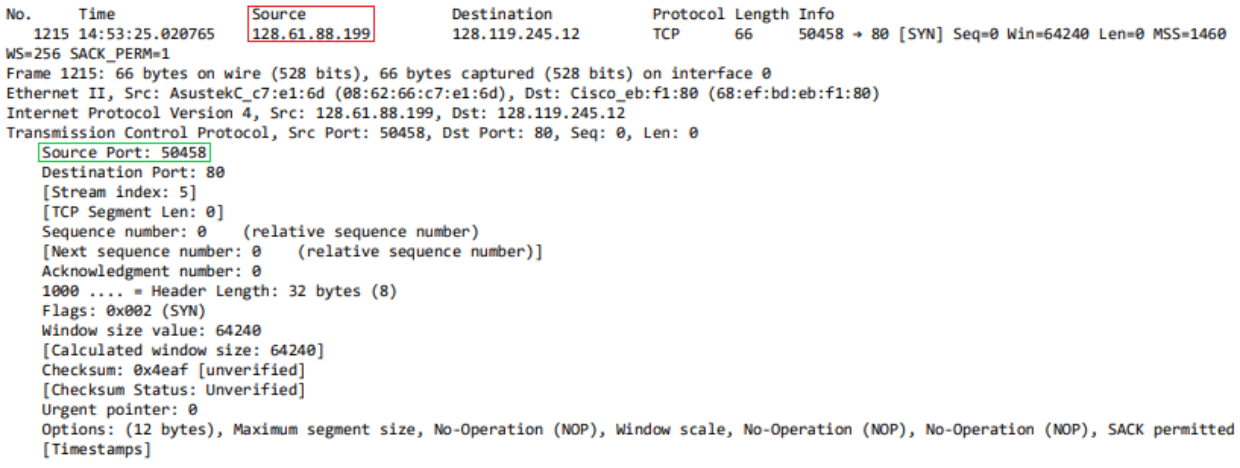
1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details 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.
    a. Client/Source IP address: 192.168.1.102 (Marked in Red)
    b. 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?
    a. Destination IP Address: 128.119.245.12 (Marked in Blue)
    b. Destination Port: 80 (Marked in Yellow)

Screen Shot for 1 and 2:



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?
    a. My Client/Sources IP address: 128.61.88.199 (Marked in Red)
    b. My Client/Sources Port: 50458 (Marked In Green)

Screen Shot for 3:



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?
    a. The Sequence Number of the TCP SYN Segment is: 0 (Marked In Red)
    b. 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:

```
No.     Time              Source                Destination           Protocol Length Info
   1215 14:53:25.020765   128.61.88.199         128.119.245.12        TCP      66     50458 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
WS=256 SACK_PERM=1
Frame 1215: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: AsustekC_c7:e1:6d (08:62:66:c7:e1:6d), Dst: Cisco_eb:f1:80 (68:ef:bd:eb:f1:80)
Internet Protocol Version 4, Src: 128.61.88.199, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 50458, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 50458
    Destination Port: 80
    [Stream index: 5]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ··········S·]
    Window size value: 64240
    [Calculated window size: 64240]
    Checksum: 0x4eaf [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
    [Timestamps]
```

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?
    a. Sequence Number of SNACK segment: 0 (Marked In Red)
    b. Value of the Acknowledgement field: 1 (Marked in Green)
    c. The server adds 1 to the initial sequence number of SYN segment that is received from the client.
    d. 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:

```
No.     Time               Source                Destination           Protocol Length Info
   1230 14:53:25.054993    128.119.245.12        128.61.88.199         TCP      66     80 → 50458 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
MSS=1460 SACK_PERM=1 WS=128
Frame 1230: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: Cisco_eb:f1:80 (68:ef:bd:eb:f1:80), Dst: AsustekC_c7:e1:6d (08:62:66:c7:e1:6d)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 128.61.88.199
Transmission Control Protocol, Src Port: 80, Dst Port: 50458, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 50458
    [Stream index: 5]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
    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
        [TCP Flags: ·····A··S·]
    Window size value: 29200
    [Calculated window size: 29200]
    Checksum: 0x4b23 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale
    [SEQ/ACK analysis]
    [Timestamps]
```

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.
    a. The sequence number of the POST command is: 1 (Marked In Red)

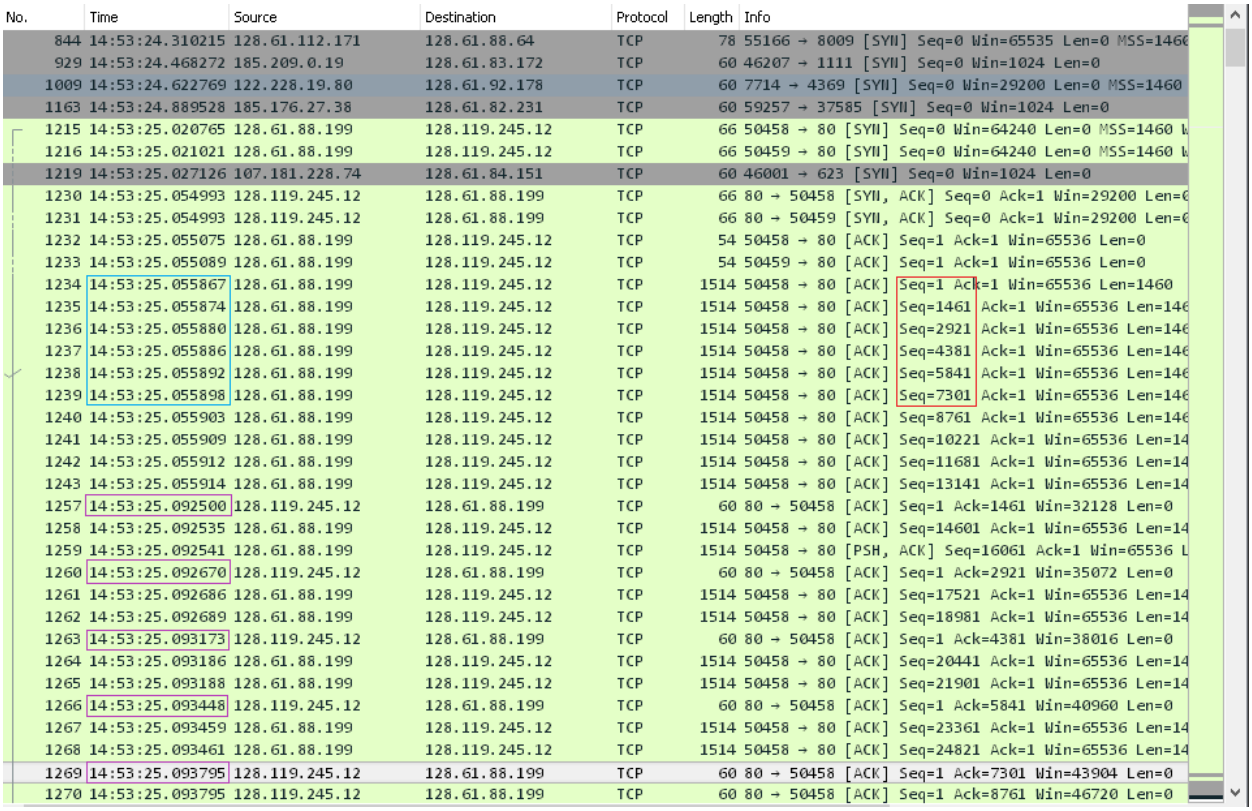
Screen Shot for 6:

```
No.     Time               Source                Destination           Protocol Length Info
   1234 14:53:25.055867    128.61.88.199         128.119.245.12        TCP      1514   50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=1460
Frame 1234: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
Ethernet II, Src: AsustekC_c7:e1:6d (08:62:66:c7:e1:6d), Dst: Cisco_eb:f1:80 (68:ef:bd:eb:f1:80)
Internet Protocol Version 4, Src: 128.61.88.199, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 50458, Dst Port: 80, Seq: 1, Ack: 1, Len: 1460
    Source Port: 50458
    Destination Port: 80
    [Stream index: 5]
    [TCP Segment Len: 1460]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1461    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
    Flags: 0x010 (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
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·····A····]
    Window size value: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0x5457 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    [SEQ/ACK analysis]
    [Timestamps]
    TCP payload (1460 bytes)
Data (1460 bytes)
0000  50 4f 53 54 20 2f 77 69 72 65 73 68 61 72 6b 2d   POST /wireshark-
0010  6c 61 62 73 2f 6c 61 62 33 2d 31 2d 72 65 70 6c   labs/lab3-1-repl
```

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 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.
    a. The first 6 sequence numbers are: 1, 1461, 2921, 4381, 5841, 7301 (Marked in Red)
    b. 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)
    c. 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)
    d.

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:

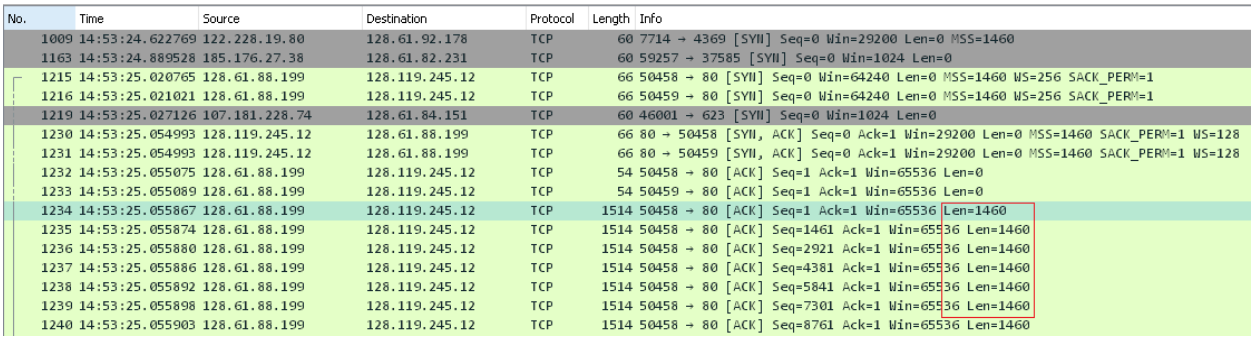


No.	Time	Source	Destination	Protocol	Length	Info
844	14:53:24.310215	128.61.112.171	128.61.88.64	TCP	78	55166 → 8009 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
929	14:53:24.468272	185.209.0.19	128.61.83.172	TCP	60	46207 → 1111 [SYN] Seq=0 Win=1024 Len=0
1009	14:53:24.622769	122.228.19.80	128.61.92.178	TCP	60	7714 → 4369 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
1163	14:53:24.889528	185.176.27.38	128.61.82.231	TCP	60	59257 → 37585 [SYN] Seq=0 Win=1024 Len=0
1215	14:53:25.020765	128.61.88.199	128.119.245.12	TCP	66	50458 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 W
1216	14:53:25.021021	128.61.88.199	128.119.245.12	TCP	66	50459 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 W
1219	14:53:25.027126	107.181.228.74	128.61.84.151	TCP	60	46001 → 623 [SYN] Seq=0 Win=1024 Len=0
1230	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50458 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
1231	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50459 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
1232	14:53:25.055075	128.61.88.199	128.119.245.12	TCP	54	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1233	14:53:25.055089	128.61.88.199	128.119.245.12	TCP	54	50459 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1234	14:53:25.055867	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=1460
1235	14:53:25.055874	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1461 Ack=1 Win=65536 Len=146
1236	14:53:25.055880	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=2921 Ack=1 Win=65536 Len=146
1237	14:53:25.055886	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=4381 Ack=1 Win=65536 Len=146
1238	14:53:25.055892	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=5841 Ack=1 Win=65536 Len=146
1239	14:53:25.055898	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=7301 Ack=1 Win=65536 Len=146
1240	14:53:25.055903	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=8761 Ack=1 Win=65536 Len=146
1241	14:53:25.055909	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=10221 Ack=1 Win=65536 Len=14
1242	14:53:25.055912	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=11681 Ack=1 Win=65536 Len=14
1243	14:53:25.055914	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=13141 Ack=1 Win=65536 Len=14
1257	14:53:25.092500	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=1461 Win=32128 Len=0
1258	14:53:25.092535	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=14601 Ack=1 Win=65536 Len=14
1259	14:53:25.092541	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [PSH, ACK] Seq=16061 Ack=1 Win=65536 L
1260	14:53:25.092670	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=2921 Win=35072 Len=0
1261	14:53:25.092686	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=17521 Ack=1 Win=65536 Len=14
1262	14:53:25.092689	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=18981 Ack=1 Win=65536 Len=14
1263	14:53:25.093173	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=4381 Win=38016 Len=0
1264	14:53:25.093186	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=20441 Ack=1 Win=65536 Len=14
1265	14:53:25.093188	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=21901 Ack=1 Win=65536 Len=14
1266	14:53:25.093448	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=5841 Win=40960 Len=0
1267	14:53:25.093459	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=23361 Ack=1 Win=65536 Len=14
1268	14:53:25.093461	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=24821 Ack=1 Win=65536 Len=14
1269	14:53:25.093795	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=7301 Win=43904 Len=0
1270	14:53:25.093795	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=8761 Win=46720 Len=0

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



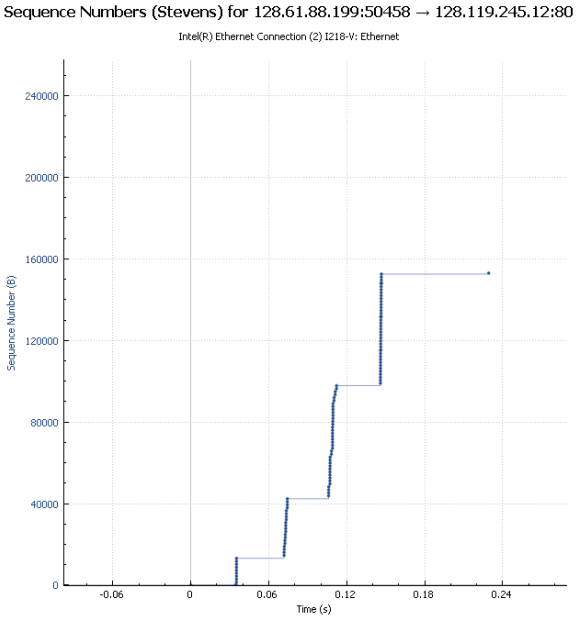
No.	Time	Source	Destination	Protocol	Length	Info
1009	14:53:24.622769	122.228.19.80	128.61.92.178	TCP	60	7714 → 4369 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
1163	14:53:24.889528	185.176.27.38	128.61.82.231	TCP	60	59257 → 37585 [SYN] Seq=0 Win=1024 Len=0
1215	14:53:25.020765	128.61.88.199	128.119.245.12	TCP	66	50458 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1216	14:53:25.021021	128.61.88.199	128.119.245.12	TCP	66	50459 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1219	14:53:25.027126	107.181.228.74	128.61.84.151	TCP	60	46001 → 623 [SYN] Seq=0 Win=1024 Len=0
1230	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50458 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
1231	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50459 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
1232	14:53:25.055075	128.61.88.199	128.119.245.12	TCP	54	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1233	14:53:25.055089	128.61.88.199	128.119.245.12	TCP	54	50459 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1234	14:53:25.055867	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=1460
1235	14:53:25.055874	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1461 Ack=1 Win=65536 Len=1460
1236	14:53:25.055880	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=2921 Ack=1 Win=65536 Len=1460
1237	14:53:25.055886	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=4381 Ack=1 Win=65536 Len=1460
1238	14:53:25.055892	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=5841 Ack=1 Win=65536 Len=1460
1239	14:53:25.055898	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=7301 Ack=1 Win=65536 Len=1460
1240	14:53:25.055903	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=8761 Ack=1 Win=65536 Len=1460

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?
    a. The minimum amount of available buffer space advertised is: 65536 (Marked in Red)
    b. There is plenty of receiver buffer space so the sender is not throttled

Screen Shot of 9:

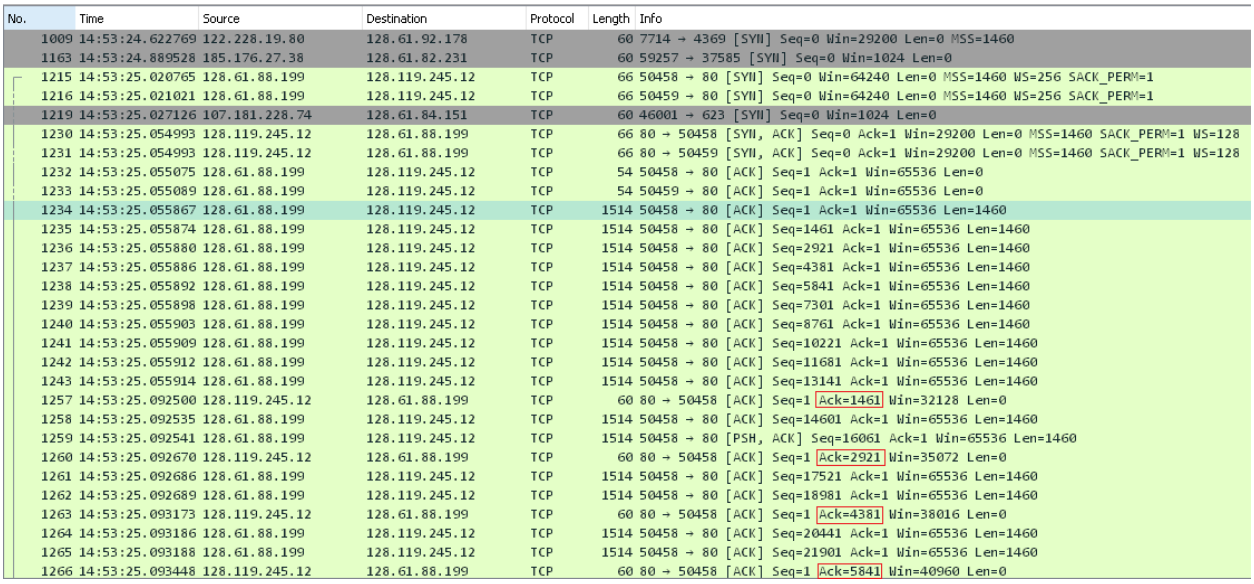
No.	Time	Source	Destination	Protocol	Length	Info
1009	14:53:24.622769	122.228.19.80	128.61.92.178	TCP	60	7714 → 4369 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
1163	14:53:24.889528	185.176.27.38	128.61.82.231	TCP	60	59257 → 37585 [SYN] Seq=0 Win=1024 Len=0
1215	14:53:25.020765	128.61.88.199	128.119.245.12	TCP	66	50458 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1216	14:53:25.021021	128.61.88.199	128.119.245.12	TCP	66	50459 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1219	14:53:25.027126	107.181.228.74	128.61.84.151	TCP	60	46001 → 623 [SYN] Seq=0 Win=1024 Len=0
1230	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50458 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
1231	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50459 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
1232	14:53:25.055075	128.61.88.199	128.119.245.12	TCP	54	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1233	14:53:25.055089	128.61.88.199	128.119.245.12	TCP	54	50459 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1234	14:53:25.055867	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=1460
1235	14:53:25.055874	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1461 Ack=1 Win=65536 Len=1460
1236	14:53:25.055880	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=2921 Ack=1 Win=65536 Len=1460
1237	14:53:25.055886	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=4381 Ack=1 Win=65536 Len=1460
1238	14:53:25.055892	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=5841 Ack=1 Win=65536 Len=1460
1239	14:53:25.055898	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=7301 Ack=1 Win=65536 Len=1460
1240	14:53:25.055903	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=8761 Ack=1 Win=65536 Len=1460
1241	14:53:25.055909	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=10221 Ack=1 Win=65536 Len=1460
1242	14:53:25.055912	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=11681 Ack=1 Win=65536 Len=1460
1243	14:53:25.055914	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=13141 Ack=1 Win=65536 Len=1460
1257	14:53:25.092500	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=1461 Win=32128 Len=0
1258	14:53:25.092535	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=14601 Ack=1 Win=65536 Len=1460
1259	14:53:25.092541	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [PSH, ACK] Seq=16061 Ack=1 Win=65536 Len=1460
1260	14:53:25.092670	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=2921 Win=35072 Len=0
1261	14:53:25.092686	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=17521 Ack=1 Win=65536 Len=1460
1262	14:53:25.092689	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=18981 Ack=1 Win=65536 Len=1460
1263	14:53:25.093173	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=4381 Win=38016 Len=0
1264	14:53:25.093186	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=20441 Ack=1 Win=65536 Len=1460
1265	14:53:25.093188	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=21901 Ack=1 Win=65536 Len=1460
1266	14:53:25.093448	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=5841 Win=40960 Len=0

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



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 250 in the text).
    a. Based on the screen shot the receiver Acknowledges 1460 bytes at a time (Marked in Red)

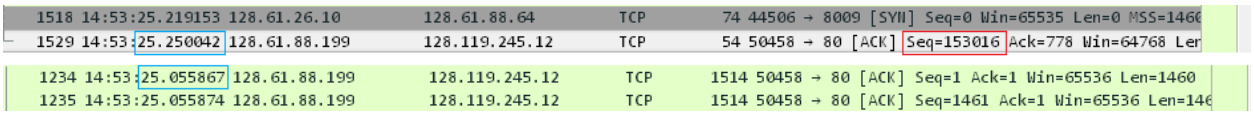
Screen Shot of 11:



No.	Time	Source	Destination	Protocol	Length	Info
1009	14:53:24.622769	122.228.19.80	128.61.92.178	TCP	60	7714 → 4369 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
1163	14:53:24.889528	185.176.27.38	128.61.82.231	TCP	60	59257 → 37585 [SYN] Seq=0 Win=1024 Len=0
1215	14:53:25.020765	128.61.88.199	128.119.245.12	TCP	66	50458 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1216	14:53:25.021021	128.61.88.199	128.119.245.12	TCP	66	50459 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1219	14:53:25.027126	107.181.228.74	128.61.84.151	TCP	60	46001 → 623 [SYN] Seq=0 Win=1024 Len=0
1230	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50458 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
1231	14:53:25.054993	128.119.245.12	128.61.88.199	TCP	66	80 → 50459 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
1232	14:53:25.055075	128.61.88.199	128.119.245.12	TCP	54	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1233	14:53:25.055089	128.61.88.199	128.119.245.12	TCP	54	50459 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1234	14:53:25.055867	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=1460
1235	14:53:25.055874	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1461 Ack=1 Win=65536 Len=1460
1236	14:53:25.055880	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=2921 Ack=1 Win=65536 Len=1460
1237	14:53:25.055886	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=4381 Ack=1 Win=65536 Len=1460
1238	14:53:25.055892	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=5841 Ack=1 Win=65536 Len=1460
1239	14:53:25.055898	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=7301 Ack=1 Win=65536 Len=1460
1240	14:53:25.055903	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=8761 Ack=1 Win=65536 Len=1460
1241	14:53:25.055909	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=10221 Ack=1 Win=65536 Len=1460
1242	14:53:25.055912	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=11681 Ack=1 Win=65536 Len=1460
1243	14:53:25.055914	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=13141 Ack=1 Win=65536 Len=1460
1257	14:53:25.092500	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=1461 Win=32128 Len=0
1258	14:53:25.092535	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=14601 Ack=1 Win=65536 Len=1460
1259	14:53:25.092541	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [PSH, ACK] Seq=16061 Ack=1 Win=65536 Len=1460
1260	14:53:25.092670	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=2921 Win=35072 Len=0
1261	14:53:25.092686	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=17521 Ack=1 Win=65536 Len=1460
1262	14:53:25.092689	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=18981 Ack=1 Win=65536 Len=1460
1263	14:53:25.093173	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=4381 Win=38016 Len=0
1264	14:53:25.093186	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=20441 Ack=1 Win=65536 Len=1460
1265	14:53:25.093188	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=21901 Ack=1 Win=65536 Len=1460
1266	14:53:25.093448	128.119.245.12	128.61.88.199	TCP	60	80 → 50458 [ACK] Seq=1 Ack=5841 Win=40960 Len=0

12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.
    a. $Throughput = \frac{Amount\ of\ Data\ Transmitted(Marked\ in\ Read)}{Time\ Elapse\ (Marked\ in\ Blue)} = \frac{153016\ bytes}{25.250042 - 25.055867}$
    $= 7880431.415\ bytes/\sec = 788.031415\ Kbytes/sec$

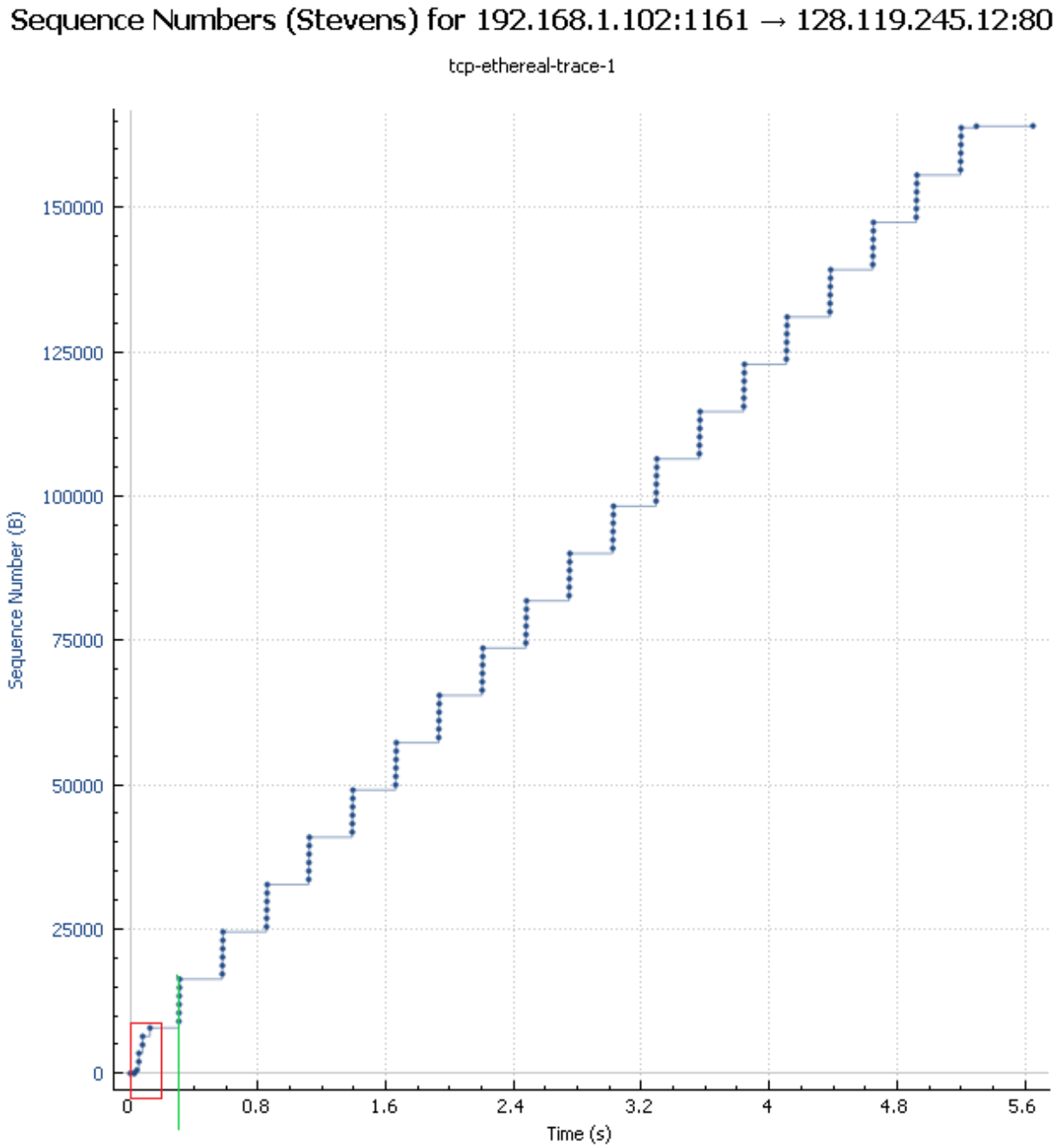
Screen Shot of 12:



1518	14:53:25.219153	128.61.26.10	128.61.88.64	TCP	74	44506 → 8009 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
1529	14:53:25.250042	128.61.88.199	128.119.245.12	TCP	54	50458 → 80 [ACK] Seq=153016 Ack=778 Win=64768 Len
1234	14:53:25.055867	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=1460
1235	14:53:25.055874	128.61.88.199	128.119.245.12	TCP	1514	50458 → 80 [ACK] Seq=1461 Ack=1 Win=65536 Len=146

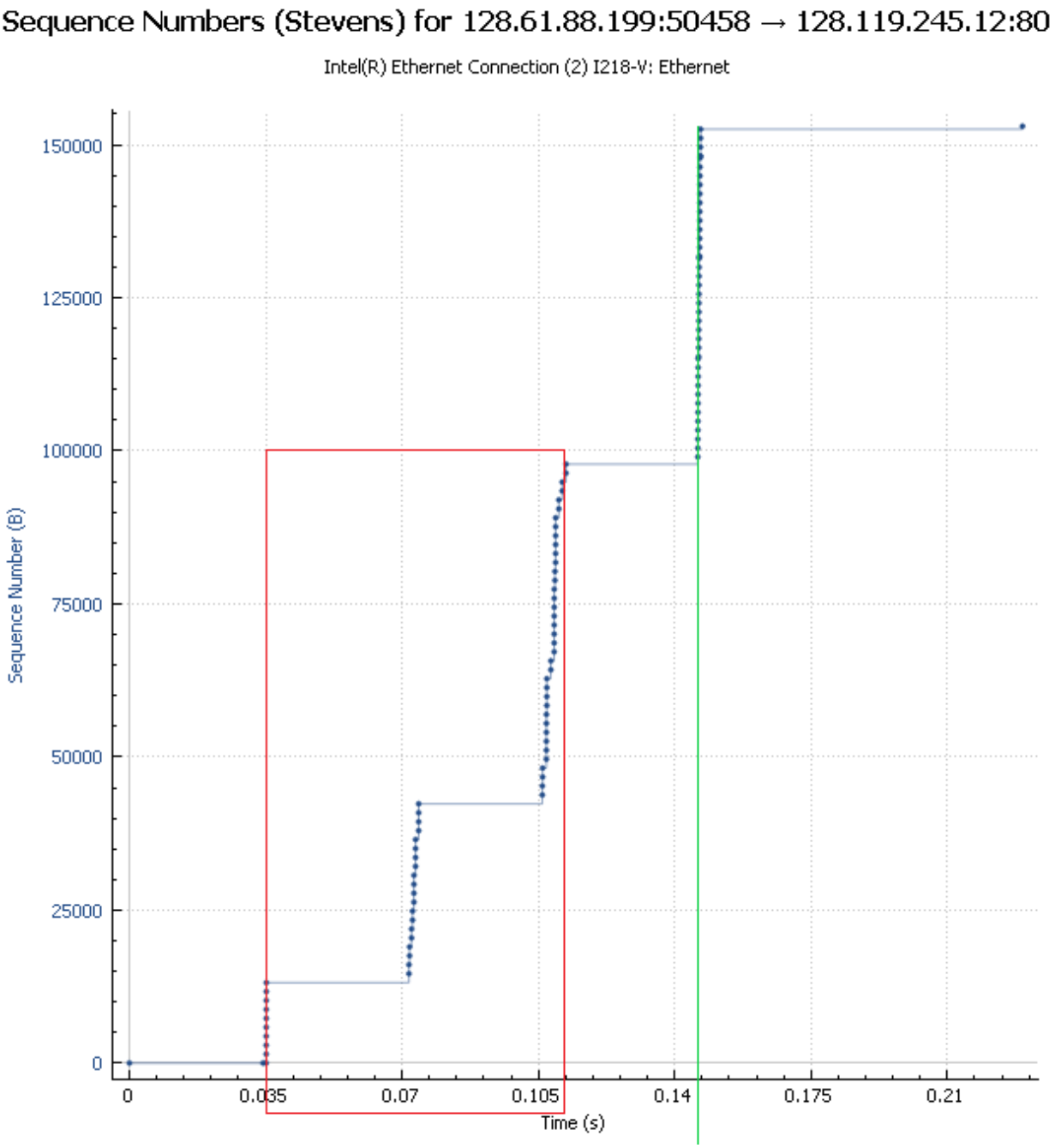
13. Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.
    a. The slow start phase looks to go from 0 sec to about 0.16 sec (Marked in Red)
    b. The congestion avoidance takes over at about about 0.32sec (Marked in Green)
    c. 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:



14. My Graph's results:
    a. The slowstart phase begins at 0.035 secs and ends at about 0.11375 sec (Marked in Red)
    b. Congestion avoidance takes over at what looks to be about 0.14875 sec (Marked in Green)
    c. 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:



15. (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?
    a. This was a design choice to help prevent redundant packet transfer.
        i. 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.

16. (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
    a. Identify the intervals of time when TCP slow start is operating.
        i. 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
    b. Identify the intervals of time when TCP congestion avoidance is operating.
        i. You can see congestion avoidance operating at [6, 16] and [17, 22] because that is where you get linear congestion window size.
    c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
        i. Segment Loss is detected by a triple duplicate ACK because if there was a timeout the window size would've dropped to 1.
    d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
        i. Segment Loss is detected by timeout because the window size drops to 1.
    e. What is the initial value of ssthresh at the first transmission round?
        i. The window seems to be set to about 32 because that is where the congestion avoidance kicks in and the slow start phase ends.
    f. What is the value of ssthresh at the 18th transmission round?
        i. 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.
    g. What is the value of ssthresh at the 24th transmission round?
        i. 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.
    h. 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

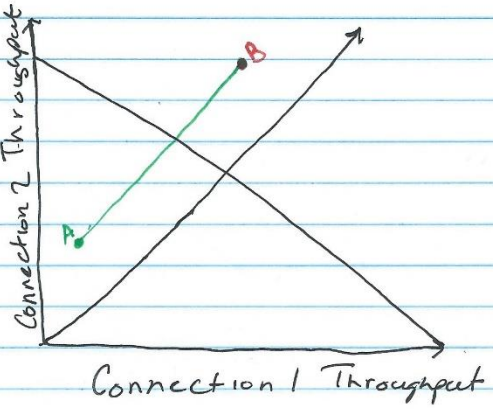
        i. Based on the table it looks like the 70th segment was sent on the 7th transmission round.
    i. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?
        i. Assuming at the 26th transmission round the threshold is at 8 after that the threshold would be set to 4 because the same reason as f.
    j. 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?
        i. The threshold would be 21 and the congestion window size would be 1 at round 19

k. 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

i. A total of 52 packets would be sent.

17. (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.



Point B is unstable and it will lead to packet loss at throughput reaches that point. Both the throughputs will additively decrease their windows causing movement along the green line towards B. At point A the sum of the throughputs will be less than the capacity of the channel and so the connections will begin to again increase in throughput again along the green line. This is going to result in movement up and down the green line an result in connection 2 getting much more of the link bandwidth. This why AIAD is not used for TCP.

18. 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?
    a. Advantage:
        i. 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.
    b. Disadvantage:
        i. 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.
    c. 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.