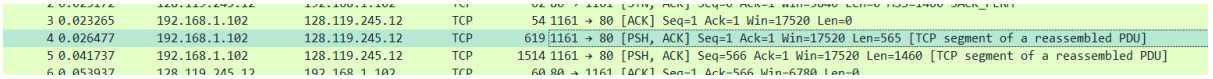
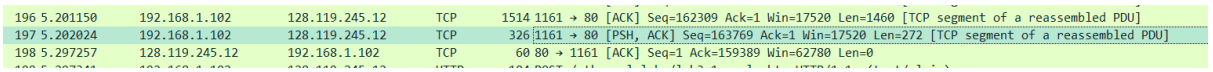
1. What are the first and last packets for the POST request?



3	0.023265	192.168.1.102	128.119.245.12	TCP	54 1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
4	0.026477	192.168.1.102	128.119.245.12	TCP	619 1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=565 [TCP segment of a reassembled PDU]
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]

here is the first



196	5.201150	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=162309 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
197	5.202024	192.168.1.102	128.119.245.12	TCP	326 1161 → 80 [PSH, ACK] Seq=163769 Ack=1 Win=17520 Len=272 [TCP segment of a reassembled PDU]
198	5.297257	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=159389 Win=62780 Len=0

and here is the last one

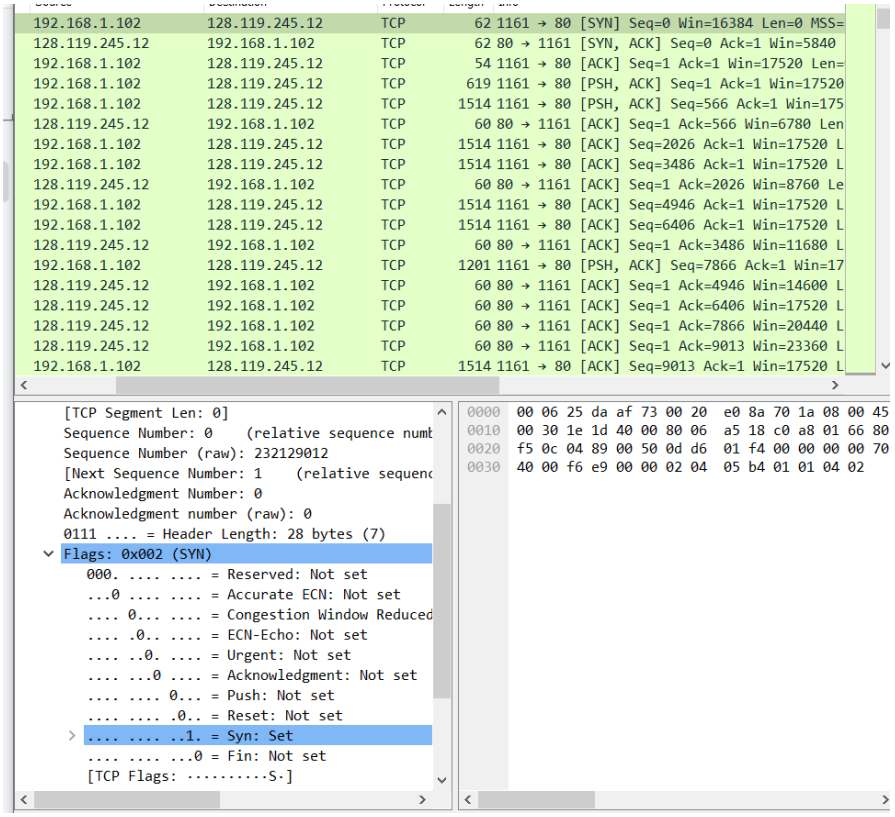
2. What is the IP address and the TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu?

   192.168.1.102:1161

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

   128.119.245.12:80

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?



192.168.1.102	128.119.245.12	TCP	62	1161 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=
128.119.245.12	192.168.1.102	TCP	62	80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840
192.168.1.102	128.119.245.12	TCP	54	1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=
192.168.1.102	128.119.245.12	TCP	619	1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=17520
192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win=175
128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len
192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=2026 Ack=1 Win=17520 L
192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=3486 Ack=1 Win=17520 L
128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 Le
192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=4946 Ack=1 Win=17520 L
192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=6406 Ack=1 Win=17520 L
128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 L
192.168.1.102	128.119.245.12	TCP	1201	1161 → 80 [PSH, ACK] Seq=7866 Ack=1 Win=17
128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=4946 Win=14600 L
128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=6406 Win=17520 L
128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=7866 Win=20440 L
128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=9013 Win=23360 L
192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=9013 Ack=1 Win=17520 L

```
[TCP Segment Len: 0]
Sequence Number: 0    (relative sequence numb
Sequence Number (raw): 232129012
[Next Sequence Number: 1    (relative sequenc
Acknowledgment Number: 0
Acknowledgment number (raw): 0
0111 .... = Header Length: 28 bytes (7)
Flags: 0x002 (SYN)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Accurate ECN: Not set
   .... 0... .... = Congestion Window Reduced
   .... .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·]
```

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

It starts with 0 and we can see here that the first packet sets to 0.

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?

```
62 1161 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 SACK_PERM
62 80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
54 1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
```

the seq=0 so it replies with 0 too

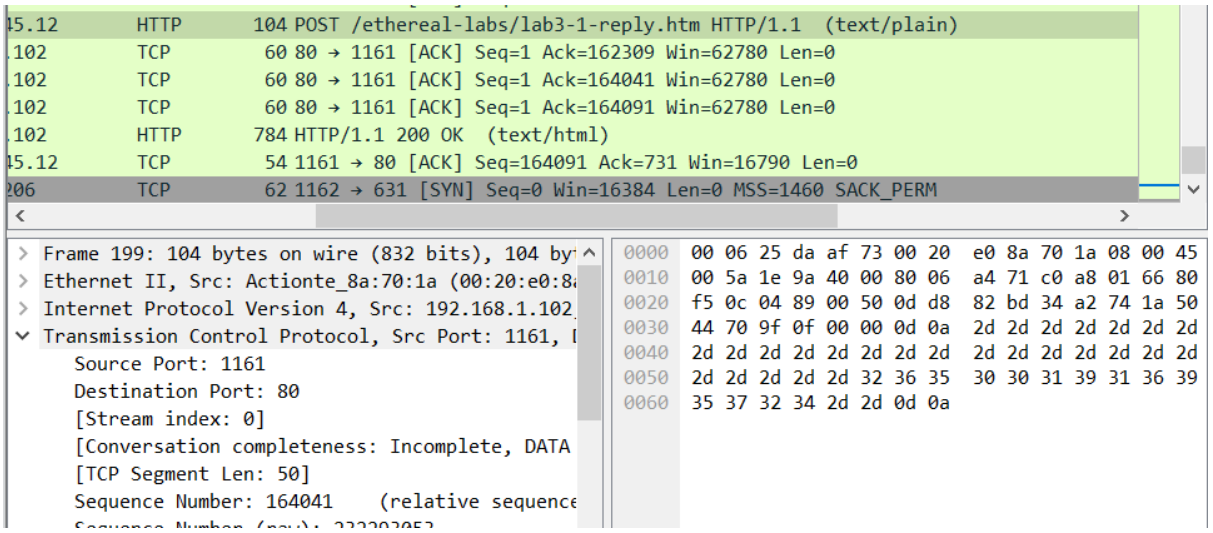
```
62 80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=
54 1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
619 1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=565 [
```

here we can see the value increments by 1.

We think that this is common practice and that is why it increments by 1 since it uses a counter to keep track of ACKs.

We know that if the SYN flag and ACK in the segment are set to 1 then we know that it is a SYN ACK.

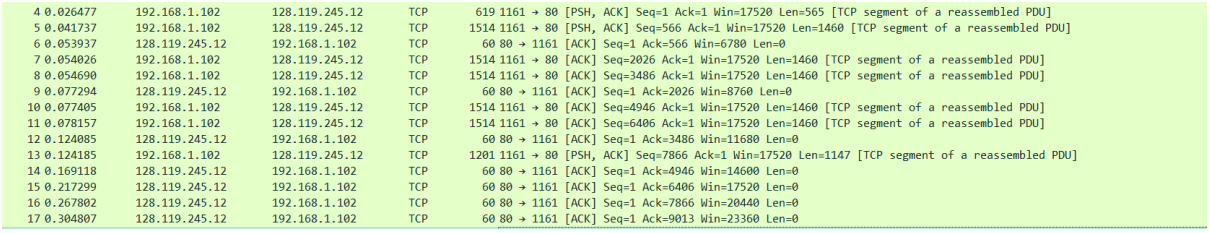
6. What is the sequence number of the TCP segment containing the HTTP POST command?



you can see the seq numbers is 164041

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 269 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 270 for all subsequent segments.



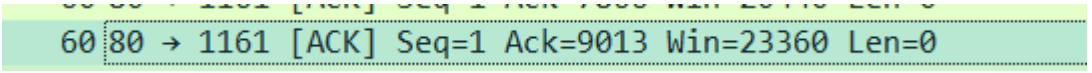
4	0.026477	192.168.1.102	128.119.245.12	TCP	619 1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=565 [TCP segment of a reassembled PDU]
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
6	0.053937	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0
7	0.054026	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=2026 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=3486 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
9	0.077294	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 Len=0
10	0.077405	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=4946 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514 1161 → 80 [ACK] Seq=6406 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
12	0.124085	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 Len=0
13	0.124185	192.168.1.102	128.119.245.12	TCP	1201 1161 → 80 [PSH, ACK] Seq=7866 Ack=1 Win=17520 Len=1147 [TCP segment of a reassembled PDU]
14	0.169118	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=4946 Win=14600 Len=0
15	0.217299	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=6406 Win=17520 Len=0
16	0.267802	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=7866 Win=20440 Len=0
17	0.304807	128.119.245.12	192.168.1.102	TCP	60 80 → 1161 [ACK] Seq=1 Ack=9013 Win=23360 Len=0

here you can see the first 6 TCP segmented files and the 6 acks received included in the file. The rtt is:

No.	RTT	EstimatedRTT (formula) (a = 0,125)
1	0,027460	0,027460
2	0,035557	0.028472125
3	0,070059	0.03367048437
4	0,114428	0.04376517382
5	0,139894	0.05578127709
6	0,189645	0.07251424245

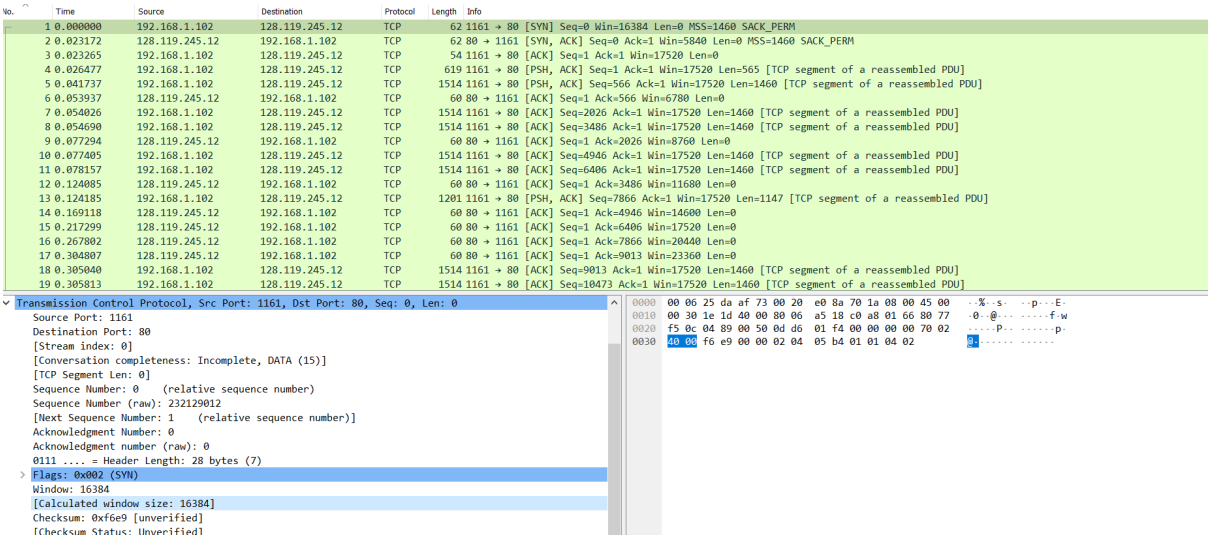
8. What is the length of each of the first six TCP segments?

We can see on the last ack is 9013 so the length would be 9012 (we start at 1)

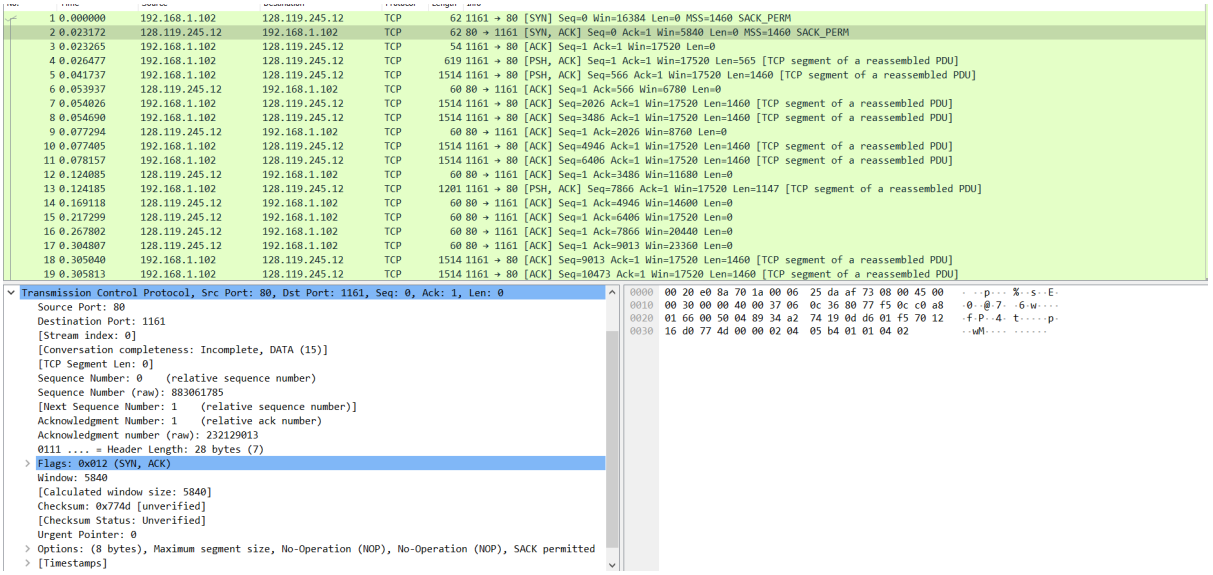


60 80 → 1161 [ACK] Seq=1 Ack=9013 Win=23360 Len=0

9. What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

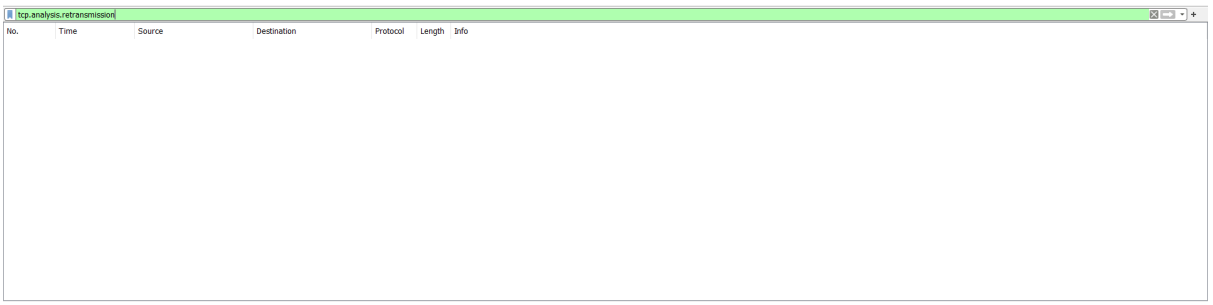


Here we can see that the minimum amount of available buffer space is 16384



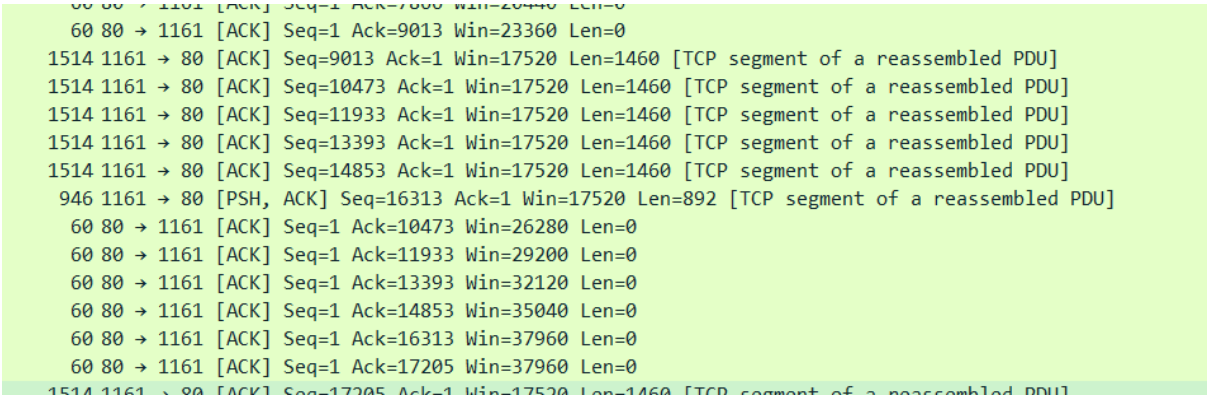
The sender shouldn't throttle because the windows size is 5840 at the lowest point. However it gets more allocated space in the buffer so this is the lowest point.

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

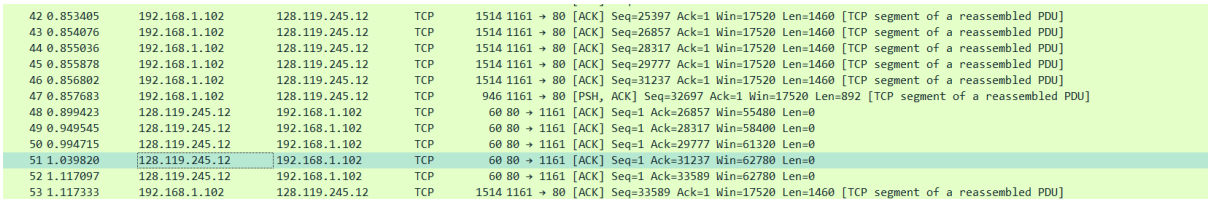


We don’t get anything retransmitted when trying to filter by “tcp.analysis.retransmission”

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 278 in the text).

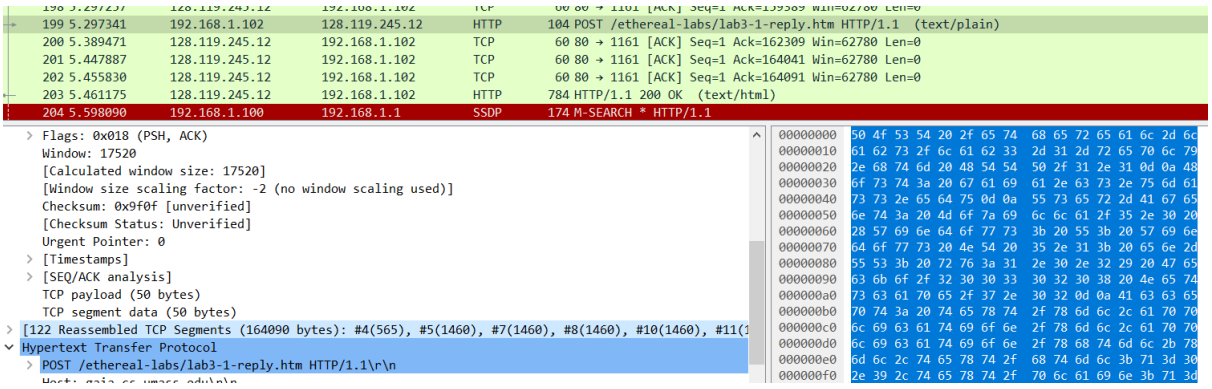


The most typical size is 1460 which we can see both on len but also on the difference between two acks. However some packets are smaller such as the 892 one you can see.



Here we can see that packet 52 is acking two packets at the same time. We can see that if you watch the ack analysis which skips one number referencing between packet 51 and 52.

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



Here can we see that the downloaded size is 164090 bytes

The difference between the first segmented packet and the last ack is:
5,45583 - 0,026477 = 5,429353 seconds.

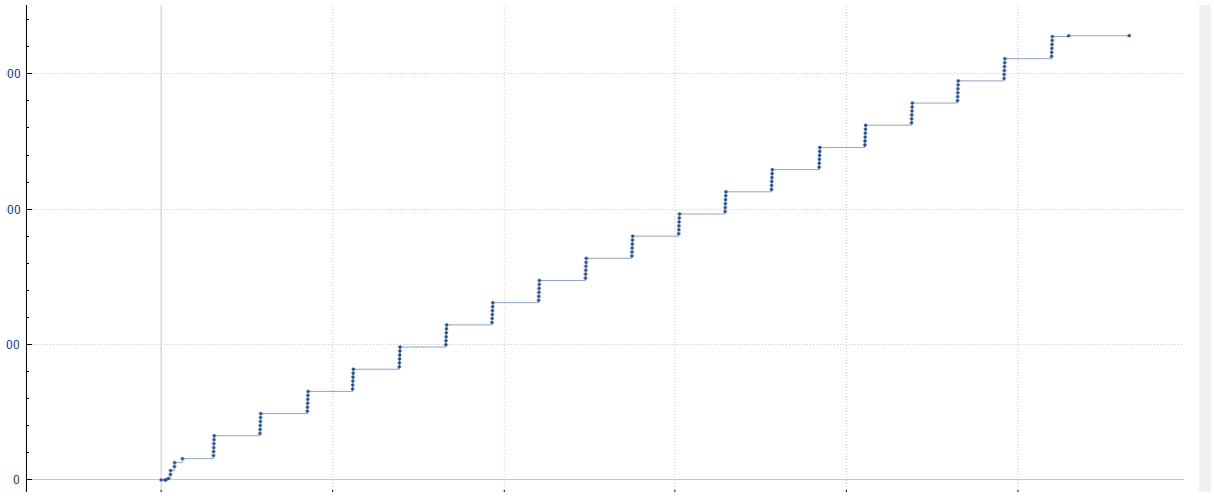
Here we can see that the throughput is 164090 / 5,429353 = 30 222,75398 bytes per second.

1-12 summary:

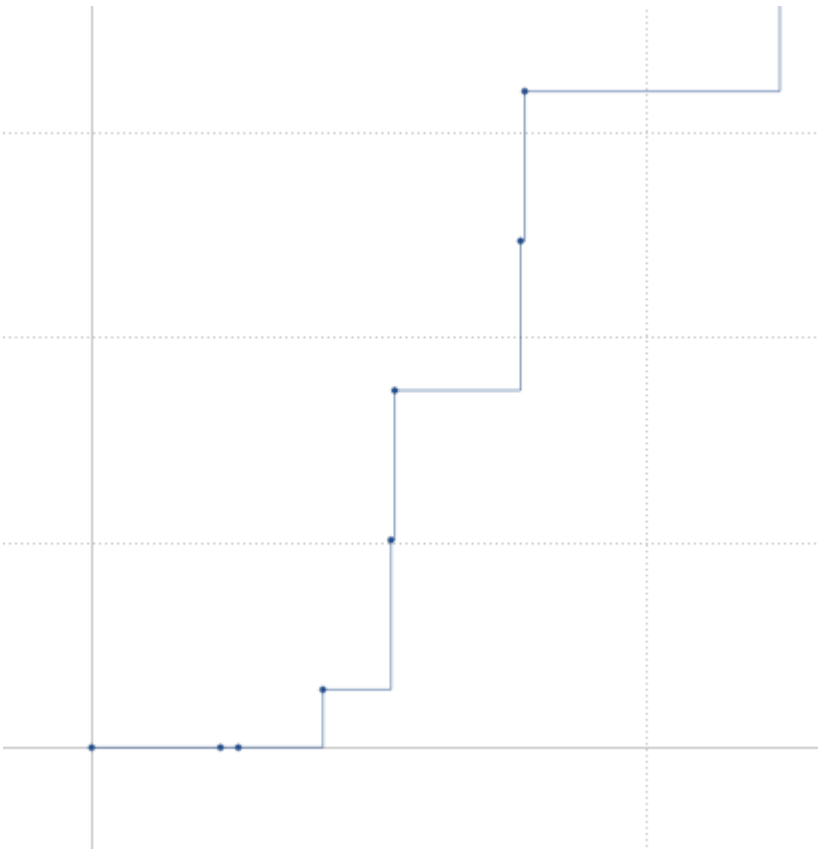
The analysis showed that the connection between the client and the server can segment a packet that is really large. We can see that ack numbers do play a role in a segmented packet. Furthermore we can see that congestion avoidance and TCP slow start is used during the lab and that is contributing to a faster internet we can use today.

The impact of RTT estimates, packet losses, and interpreted packet loss events can greatly affect the performance of a TCP connection. Accurate RTT estimation is crucial for maintaining a stable and efficient connection. Packet losses can lead to retransmissions, causing delays and reducing throughput. Interpreting packet loss events can help identify potential network issues and improve network performance. In this trace, we did not find any retransmissions, which suggests a stable connection.

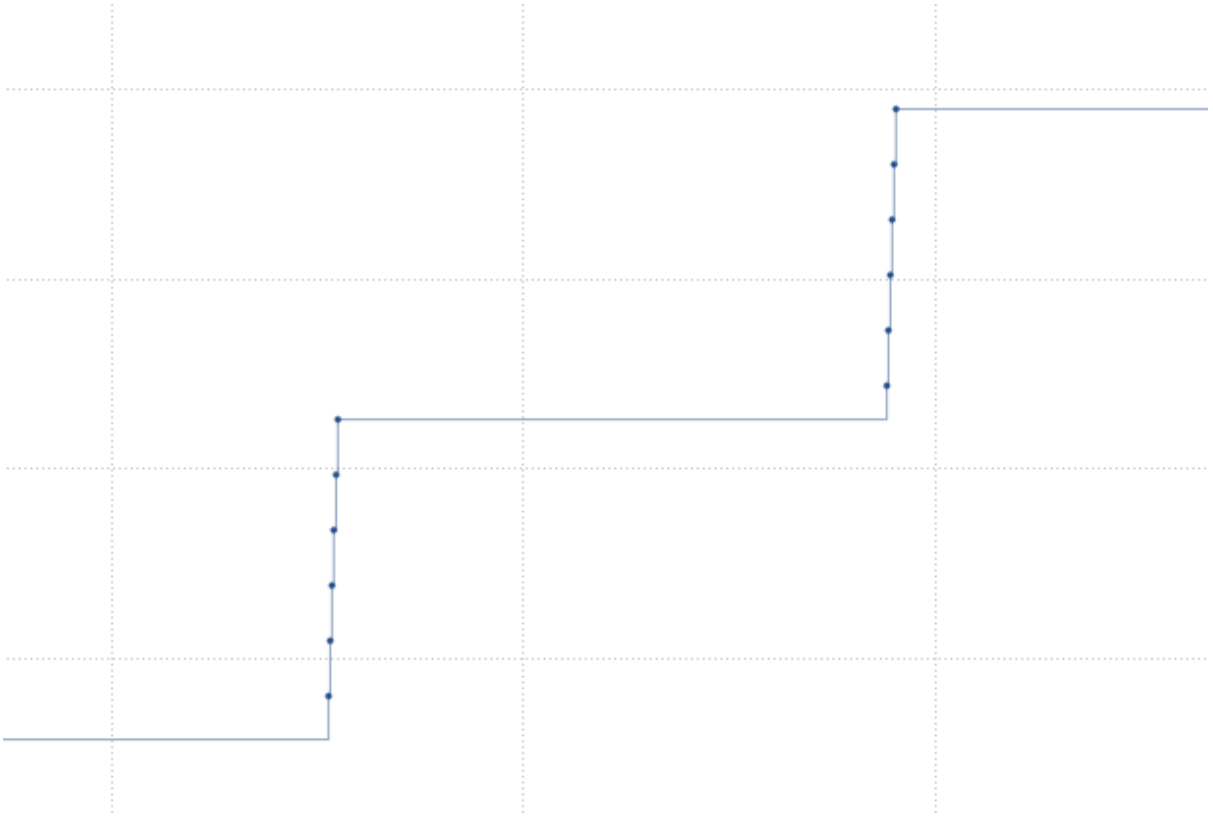
13. Use the *Time-Sequence-Graph (Stevens)* plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify if and where TCP's *slow start* phase begins and ends, as well as if and where *congestion avoidance* takes over?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



We can see the, the tcp slow start starts directly with a slow stream of packets. second plateau which goes from around 0,25 and around 0,3 is when the tcp slow start phases ends and we can see that the bursts of packets gets larger and larger.



We can see the congestion avoidance kicks in where it starts to continue to burst packets.



14. Explain the relationship between (i) the congestion window (cwnd), (ii) the receiver advertised window (rwnd), (iii) the number of unacknowledged bytes, and (iv) the effective window at the sender (i.e., the window effectively limiting the data transmission).

The cwnd, rwnd, number of unacknowledged bytes, and effective window are all interdependent and work together to control the amount of data that can be transmitted over a TCP connection. By adjusting the cwnd and rwnd dynamically based on network conditions, the sender can ensure that it does not overload the network, while still maintaining a high data transfer rate. The number of unacknowledged bytes and the effective window provide additional checks to prevent the sender from overwhelming the receiver with data.

15. Is it **generally** possible to find the congestion window size (cwnd) and how it changes with time, from the captured trace files? If so, please explain how. If not, please explain when and when not. Motivate your answer and give examples.

The cwnd value can be found in the TCP header of each packet, which can be inspected in the trace file. Wireshark has a built-in feature that calculates and graphs the cwnd values over time, which allows you to visualize how the congestion window changes during a TCP connection.

16. What is the throughput of each of the connections in bps (bits per second)? What is the total bandwidth of the host on which the clients are running? Discuss the TCP fairness for this case.

Total size/ Duration = throughput (bps)	
2535059	
2546529	
2575234	Total bandwidth:
2550558	10 207 380

The TCP fairness is pretty fair in our opinion because of the very similar throughput. So it is fair between the connections. The RTT is the same for all the connections which makes it even more fair.

17. What is the throughput of each of the connections in bps (bits per second)? What is the total bandwidth of the host on which the clients are running? Discuss the TCP fairness for this case.

Total size/ Duration = throughput (bps)	
23 228 367	
15 644 074	
13 501 738	
12 478 984	
9 654 284	
6 279 528	
5 843 994	
3 841 144	Total Bandwidth:
3 486 446	93 958 559

The TCP fairness isn't fair. You can see that between the first two connections which differ by almost 8 million bps. You can also see on the RTT that there is a huge difference between the connections.

18. Discuss the TCP fairness for this case. How does it differ from the previous cases, and how is it affected by the use of BitTorrent?

Total size/ Duration = throughput (bps)	
15 013 949	
12 473 887	
8 750 427	
8 827 589	
7 441 676	
7 019 190	
6 794 729	
5 547 784	
5 387 831	Total bandwidth:
5 273 158	82 530 220

Here we can see that the TCP fairness isn’t that great either, however this is in a torrent situation where you both download and upload files. In some cases it can be a large travel time for one packet to reach each user. Therefore it is not fair but you can’t really do anything about it. However you also need to upload files which take bandwidth which affects the throughput. You also need to take in account that RTT varies much because of the different packets being sent everywhere.