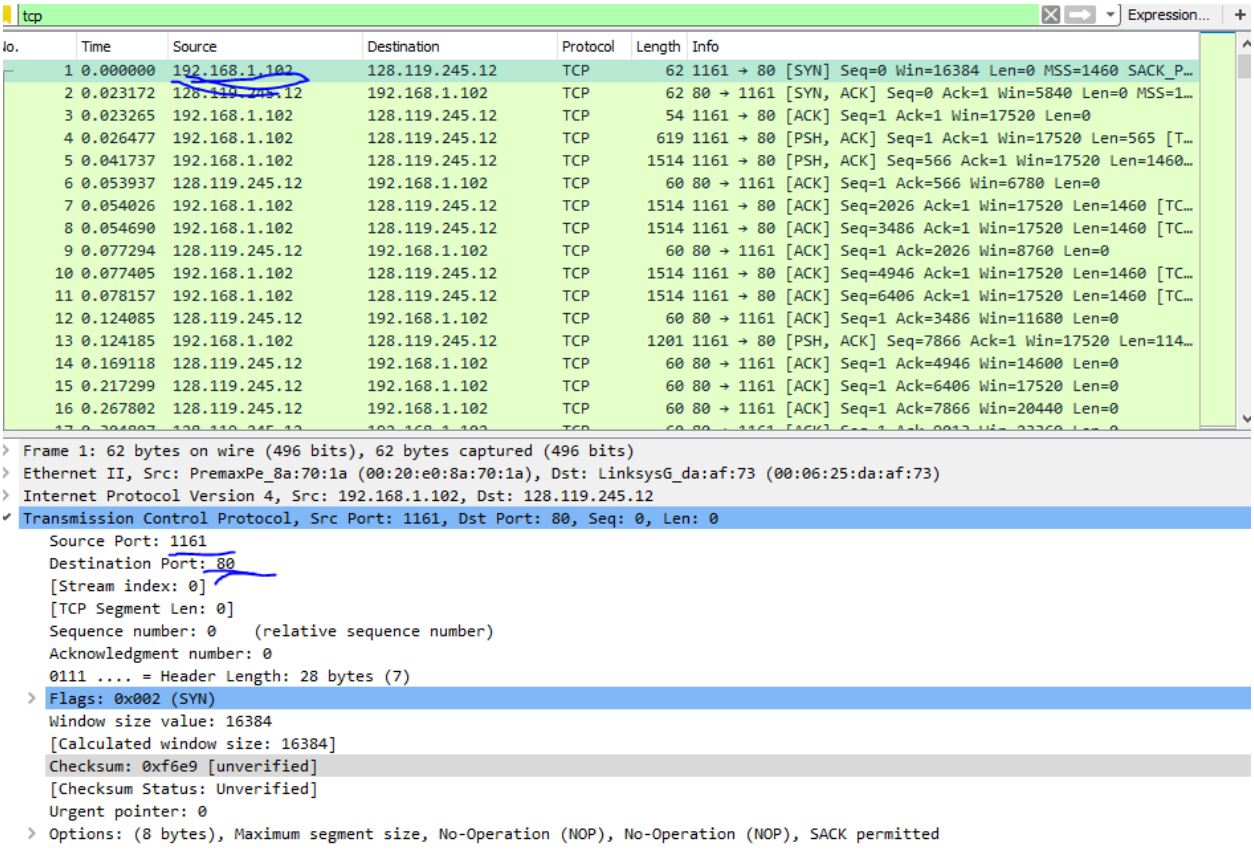
Valeriy Kutsar Wirehark Lab 3

1. IP address is 192.168.1.102 the TCP port number is 1161.
2. IP address of gaia.cs.umass.edu is 128.119.245.12 and it is sending and receiving packets from source number 80.

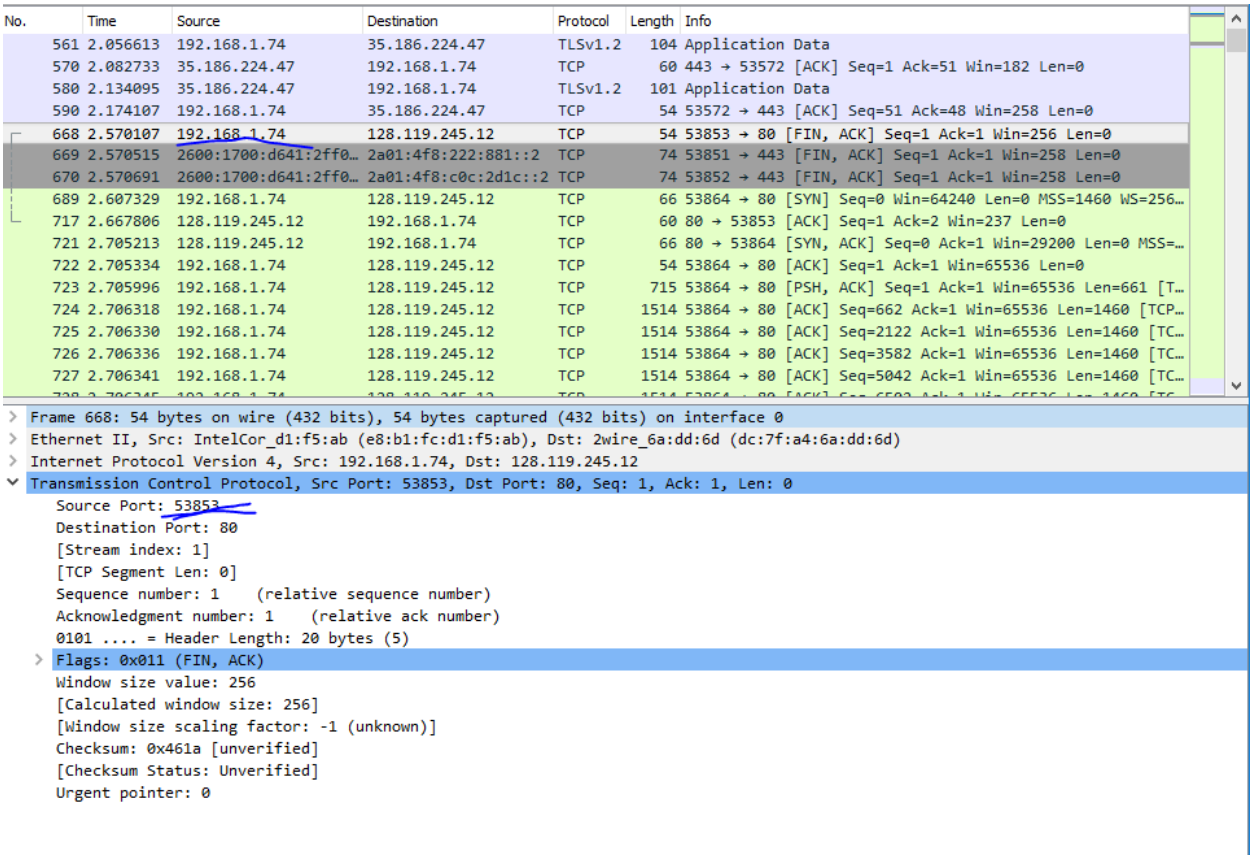


tcp

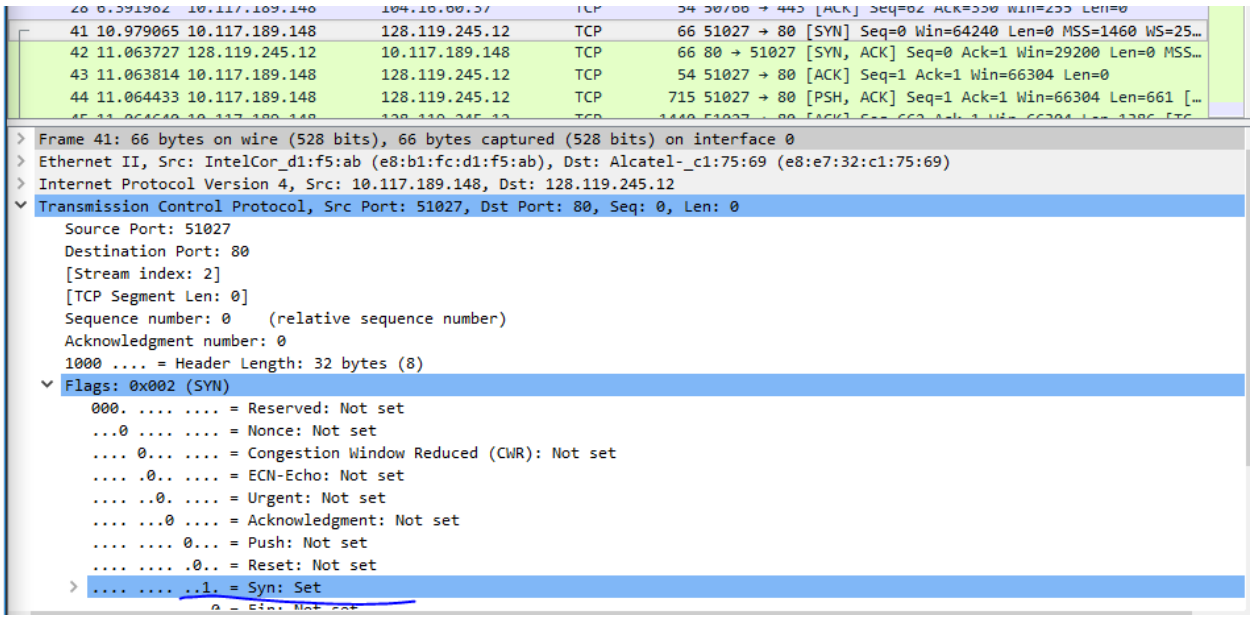
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.102	128.119.245.12	TCP	62	1161 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 SACK_P…
2	0.023172	128.119.245.12	192.168.1.102	TCP	62	80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1…
3	0.023265	192.168.1.102	128.119.245.12	TCP	54	1161 → 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 [T…
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…
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 [TC…
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=3486 Ack=1 Win=17520 Len=1460 [TC…
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 [TC…
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=6406 Ack=1 Win=17520 Len=1460 [TC…
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=114…
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

```
Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
Ethernet II, Src: PremaxPe_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, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 1161
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 0
    0111 .... = Header Length: 28 bytes (7)
    Flags: 0x002 (SYN)
    Window size value: 16384
    [Calculated window size: 16384]
    Checksum: 0xf6e9 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    Options: (8 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted
```

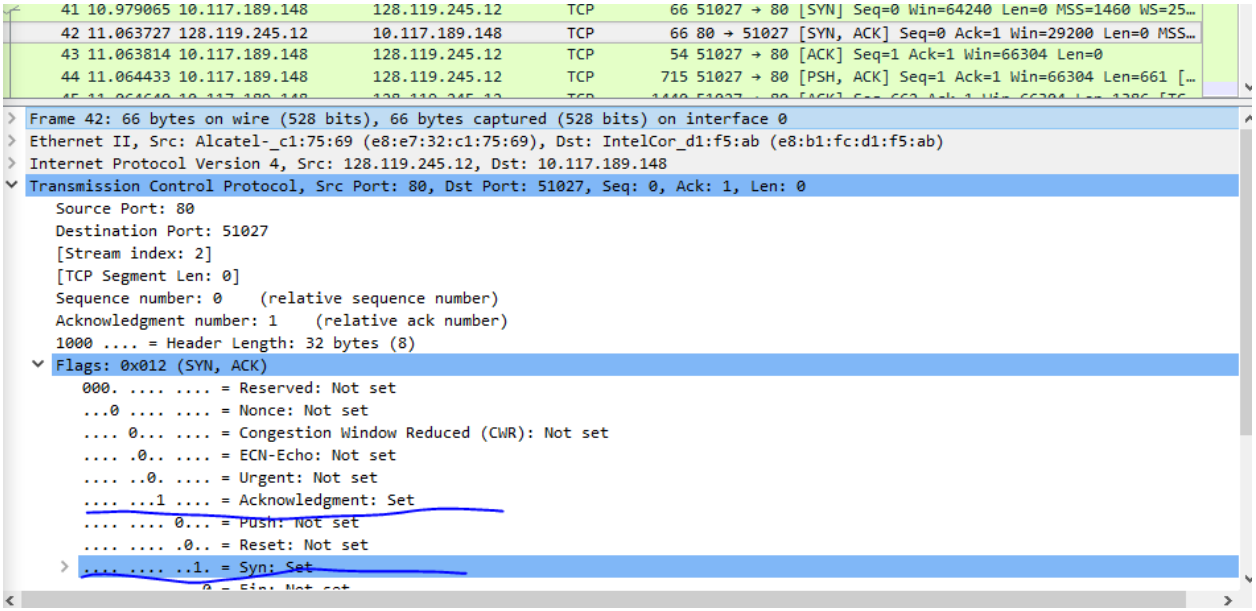
3. From my own trace, my IP address is 192.168.1.74 and the source port number is 53853.



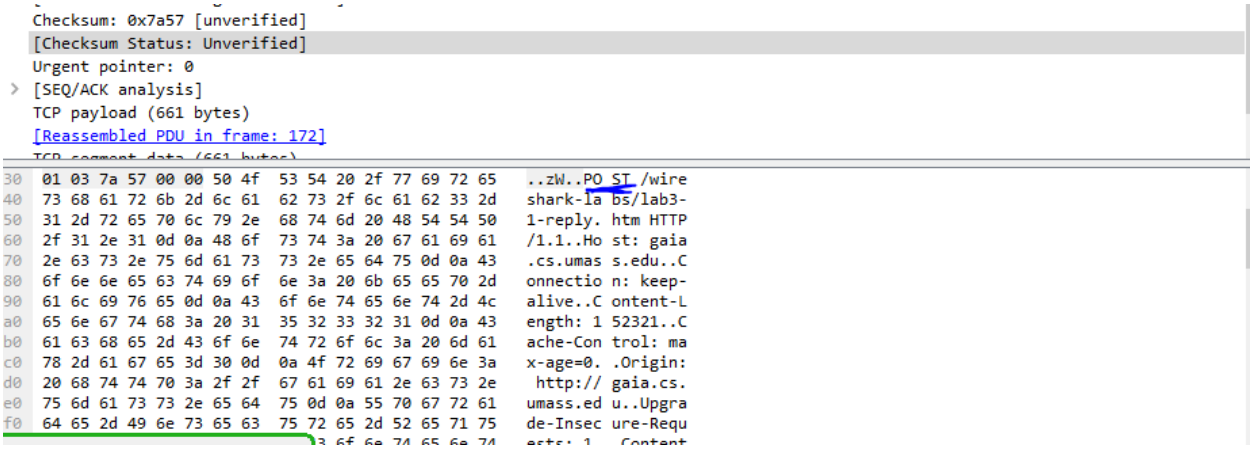
4. The sequence number is 0. The ack number acknowledges the number as part of the SYN segment. The SYN flag is set to 1 and it indicates that this segment is a SYN segment.



5. The sequence number sent from gaia is 0. The acknowledgement value is 1. It determined it by adding 1 to the initial sequence number of SYN segment from the client computer. The SYN flag and ACK flag in the segment are set to 1 and they indicate that this segment is a SYNACK segment.



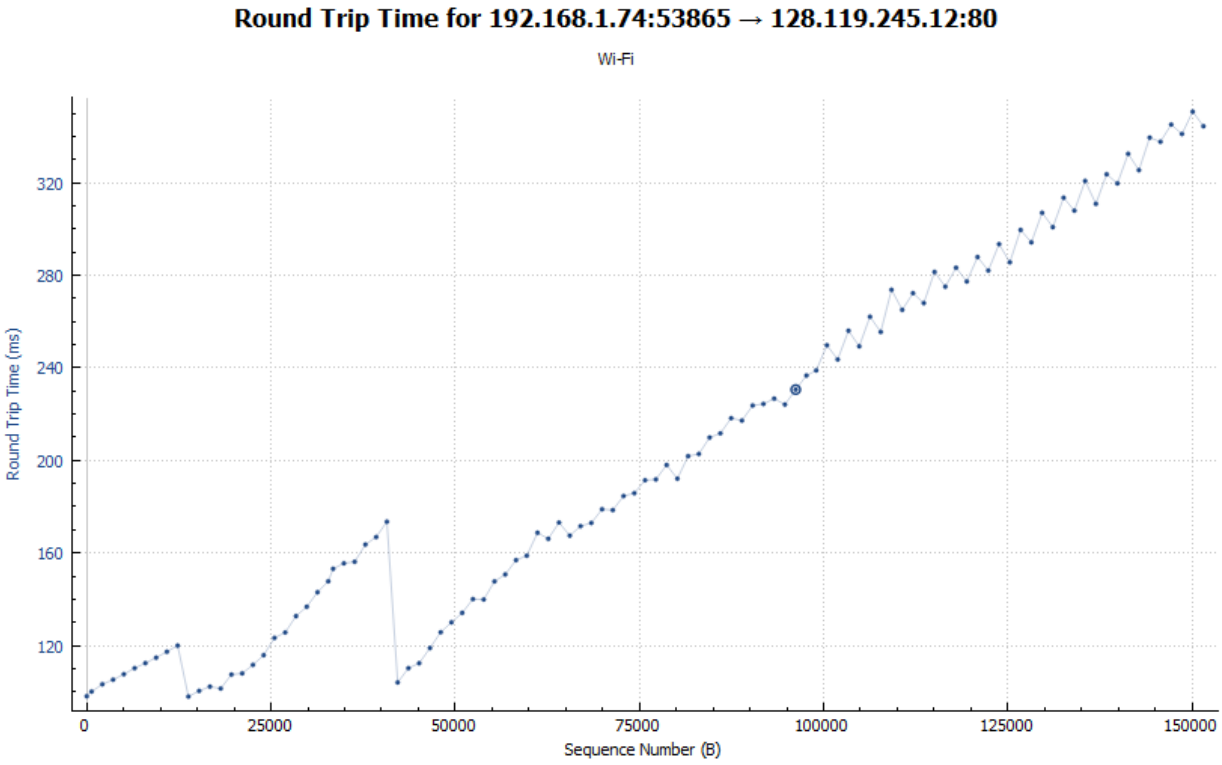
6. The post has a sequence number of 1.



7. Segment 1 seq #: 1
   Segment 2 seq#:662
   Segment 3 seq#:2122
   Segment 4 seq#:3582
   Segment 5 seq#:5042
   Segment 6 seq#:6502

Segment #	Sent Time	ACK received time	RTT
1	2.705996	2.759786	.09528
2	2.706318	2.804927	.098931
3	2.706330	2.806412	.100094

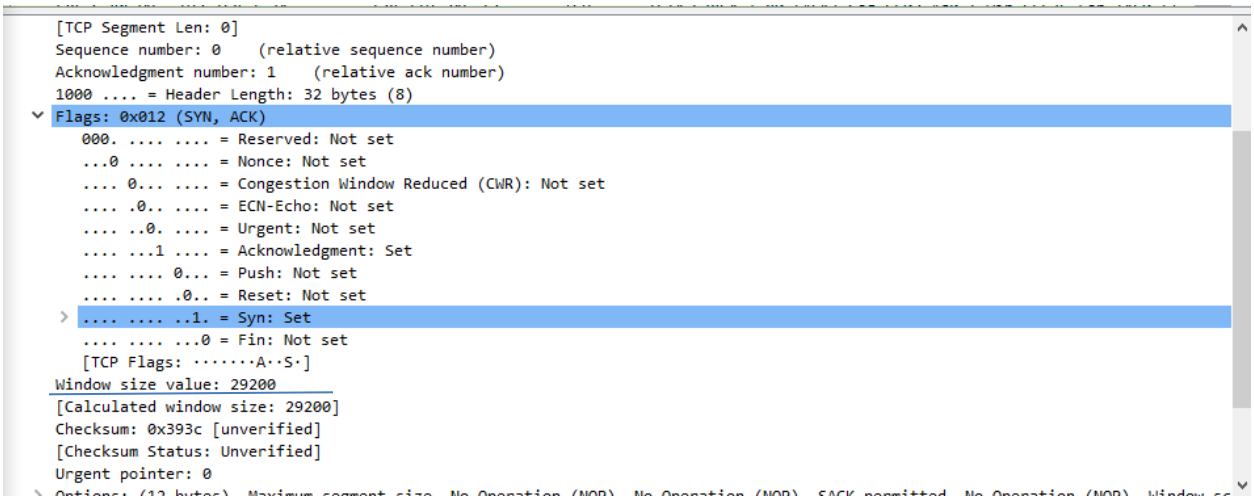
4	2.706336	2.809844	.103514
5	2.706341	2.81141	.105074
6	2.706345	2.813538	.107197



8. Length of first TCP segment with POST: 661 bytes. Length of each of the other 5 TCP segments: 1460 bytes.

9. The minimum amount of buffer space(receiver window) advertised at gaia for the entire trace is 29200 bytes, which shows in the first ACK from the server. This

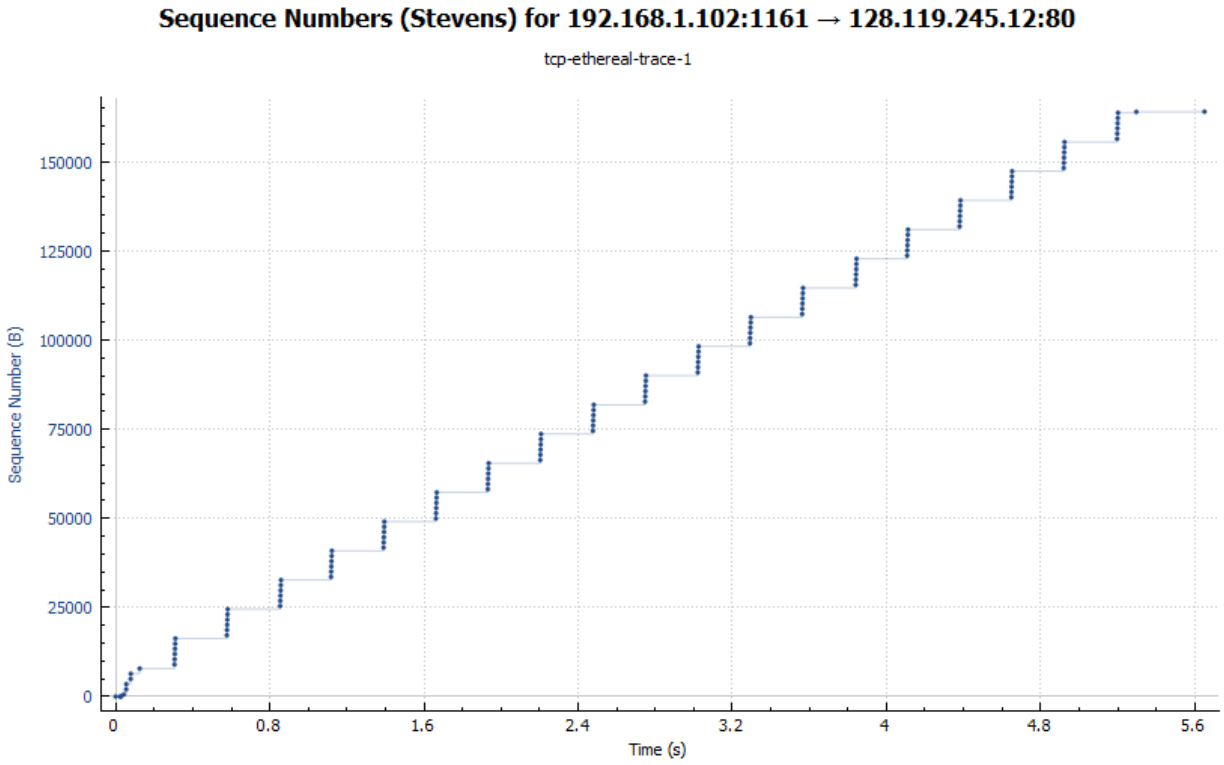
receiver window grows steadily until a maximum buffer size of 64768 bytes. The sender is never throttled due to lacking of receiver buffer space by inspecting this trace.



10. No packets we retransmitted because there were not any duplicate ACK's.
11. The difference between the acknowledged sequence numbers of to consecutive ACK's indicates the data received by the server between these two ACK's. By looking at the segments, we can see that some segments acknowledged data with 2 times the regular data.
12. The throughput can be computed as the ratio between the total amount of data and the total transmission time. The total amount data transmitted can be computed by the difference between the sequence number of the first TCP and the acknowledged sequence number of the last ACK(152983-1 = 152982). The whole transmission time is the difference of the time instant of the first TCP segment(3.472449 – 2.705996 = .766456). Therefore the computed throughput for my trace was 152982/.766453) which gives 199597.366 Bytes/sec.

13. From the chart we can see that the amount of data being sent increased in the beginning, but did not exceed a certain limit. We cannot exactly know where the slow

start phase and the start of the congestion avoidance phase for this trace because the sender is not sending data aggressively enough for the need to use congestion avoidance. Before it receives the acknowledgement for the whole block of data, the application will not send more data and that shows how before the end of the slow start phase, the application stops sending data for a short time.



14.

Congestion control did not need take over. The TCP sender is not sending data aggressively enough to push to the congestion state. Before the sender receives acknowledgement of the sent data, it stops sending data. The rate of the transfer grows exponentially, and if there was a need for congestion control, there would be drops in the graph, of which there are none in this trace. Same as the wireshark sample graph, the application stopped sending data for a short time before it received the whole block of data, and then continued sending data