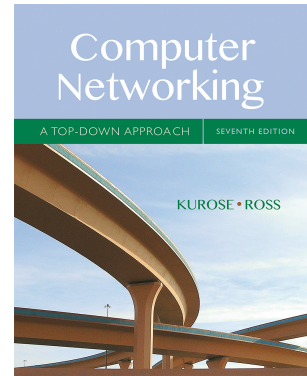


Wireshark Lab: TCP

SOLUTION

Supplement to *Computer Networking: A Top-Down Approach*, 7th ed., J.F. Kurose and K.W. Ross

© 2005-2016, J.F Kurose and K.W. Ross, All Rights Reserved



The answers below are based on the trace file *tcp-ethereal-trace-1* in in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>

TCP Basics

Answer the following questions for the TCP segments:

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

Solution: Client computer (source)

IP address: 192.168.1.102

TCP port number: 1161

Destination computer: gaia.cs.umass.edu

IP address: 128.119.245.12

TCP port number: 80

3. If you did this problem on your own computer, you'll have your own solution

| No. | Time | Source | Destination | Protocol | Info |
|-----|----------|----------------|----------------|----------|--|
| 1 | 0.000000 | 192.168.1.102 | 128.119.245.12 | TCP | 1161 > http [SYN] Seq=0 Ack=0 Win=16384 Len=0 MSS=1460 |
| 2 | 0.023172 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 |
| 3 | 0.023265 | 192.168.1.102 | 128.119.245.12 | TCP | 1161 > http [ACK] Seq=1 Ack=1 Win=17520 Len=0 |
| 4 | 0.026477 | 192.168.1.102 | 128.119.245.12 | HTTP | POST /ethereal-labs/lab3-1-reply.htm HTTP/1.1 |
| 5 | 0.041737 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |
| 6 | 0.053937 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0 |
| 7 | 0.054026 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |
| 8 | 0.054690 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |
| 9 | 0.077294 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [ACK] Seq=1 Ack=2026 Win=8760 Len=0 |
| 10 | 0.077405 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |
| 11 | 0.078157 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |
| 12 | 0.124085 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [ACK] Seq=1 Ack=3486 Win=11680 Len=0 |
| 13 | 0.124185 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |
| 14 | 0.169118 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [ACK] Seq=1 Ack=4946 Win=14600 Len=0 |
| 15 | 0.217299 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [ACK] Seq=1 Ack=6406 Win=17520 Len=0 |
| 16 | 0.267802 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [ACK] Seq=1 Ack=7866 Win=20440 Len=0 |
| 17 | 0.304807 | 128.119.245.12 | 192.168.1.102 | TCP | http > 1161 [ACK] Seq=1 Ack=9013 Win=23360 Len=0 |
| 18 | 0.305040 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |
| 19 | 0.305813 | 192.168.1.102 | 128.119.245.12 | HTTP | Continuation or non-HTTP traffic |

* Frame 1 (62 bytes on wire, 62 bytes captured)
 * Ethernet II, Src: Actionte_8a:70:1a (00:20:e0:8a:70:1a), Dst: 192.168.1.1 (00:06:25:da:af:73)
 * Internet Protocol, 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: http (80), Seq: 0, Ack: 0, Len: 0

```

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: "Z:\course\PolyU\CS 684\ethereal-traces\tcp-ethereal-trace-1" 177 KB | P: 213 D: 213 M: 0

Figure 1: IP addresses and TCP port numbers of the client computer (source) and gaia.cs.umass.edu

- 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?

Solution: Sequence number of the TCP SYN segment is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu. The value is 0 in this trace.

The SYN flag is set to 1 and it indicates that this segment is a SYN segment.

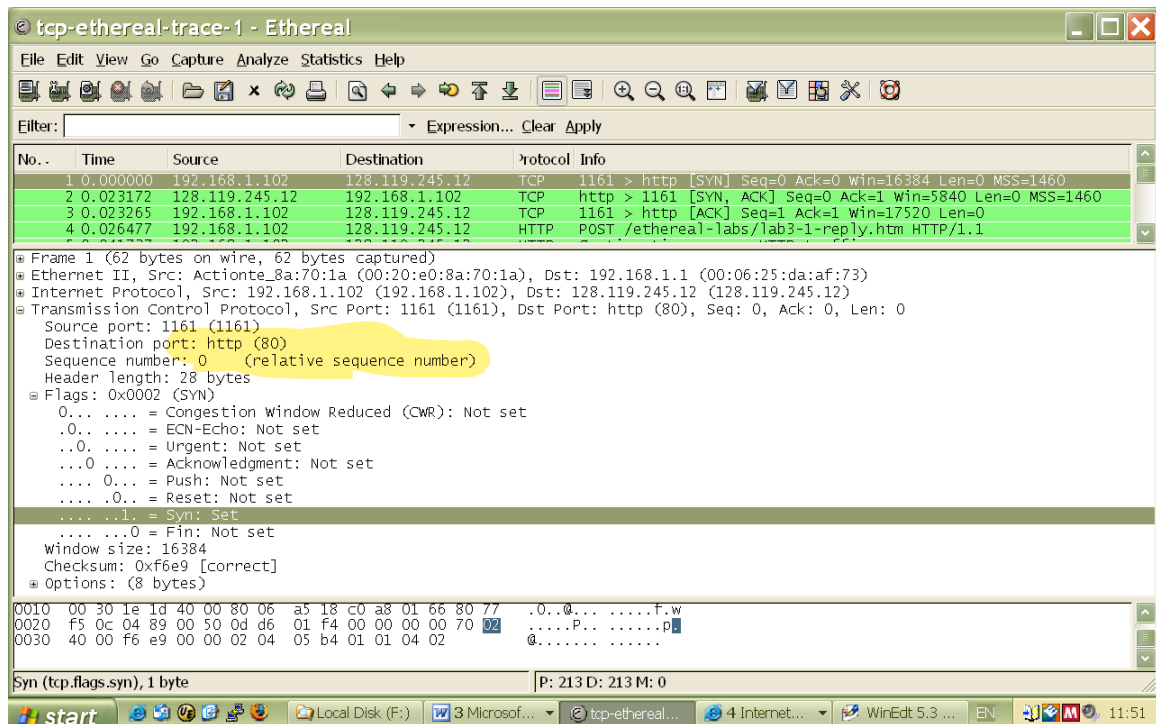


Figure 2: Sequence number of the TCP SYN segment

- 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?

Solution: Sequence number of the SYNACK segment from *gaia.cs.umass.edu* to the client computer in reply to the **SYN has the value of 0** in this trace.

The value of the ACKnowledgement field in the SYNACK segment is 1. The value of the ACKnowledgement field in the SYNACK segment is determined by *gaia.cs.umass.edu* by adding 1 to the initial sequence number of SYN segment from the client computer (i.e. the sequence number of the SYN segment initiated by the client computer is 0.).

The SYN flag and Acknowledgement flag in the segment are set to 1 and they indicate that this segment is a SYNACK segment.

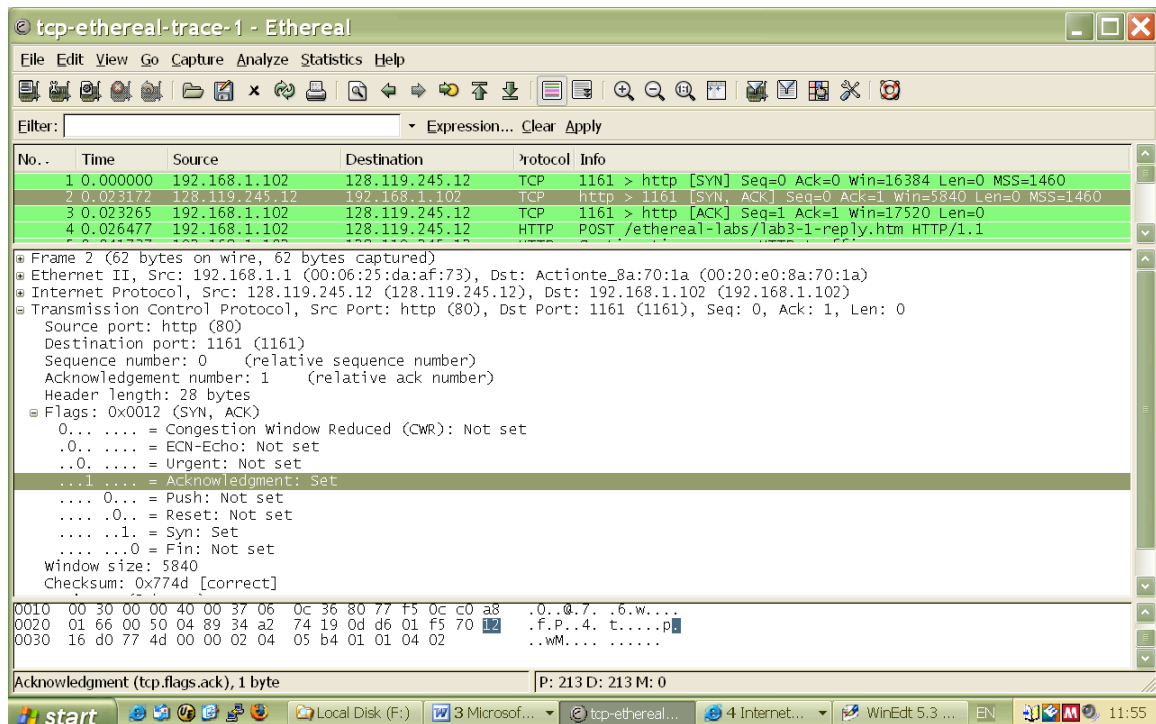


Figure 3: Sequence number and Acknowledgement number of the SYNACK segment

- 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.

Solution: No. 4 segment is the TCP segment containing the HTTP POST command. The sequence number of this segment has the value of 1.

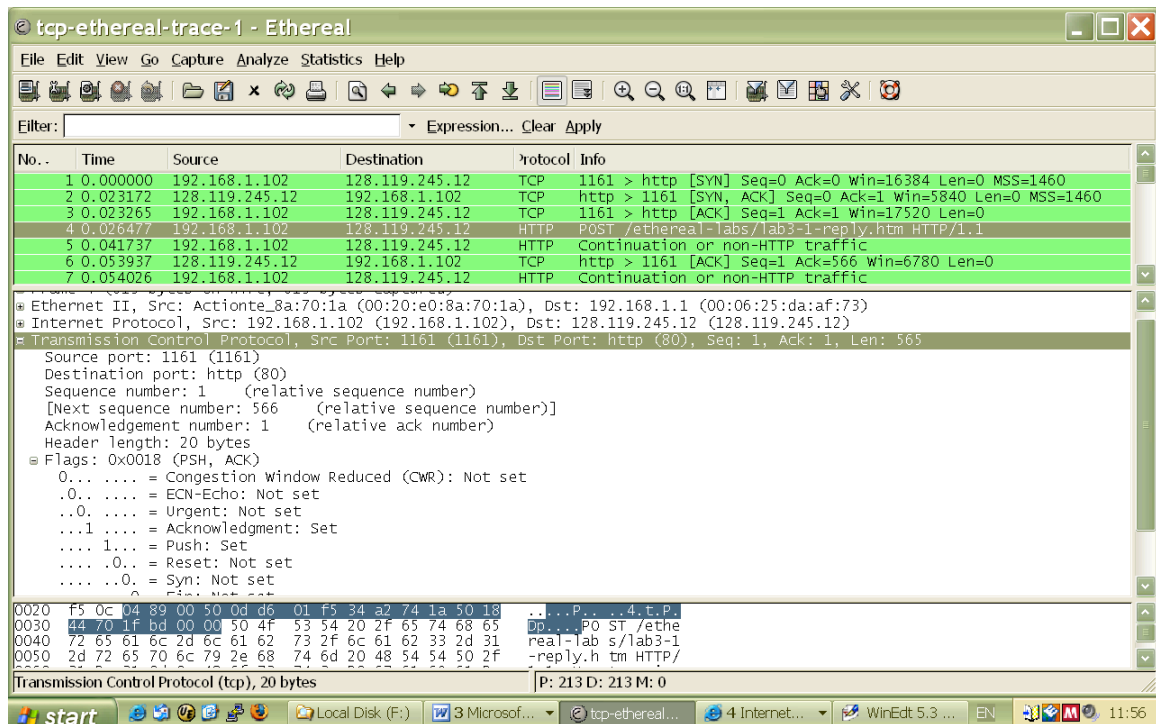


Figure 4: Sequence number of the TCP segment containing the HTTP POST command

- 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 page 237 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 237 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.

Solution: The HTTP POST segment is considered as the first segment. Segments 1 – 6 are No. 4, 5, 7, 8, 10, and 11 in this trace respectively. The ACKs of segments 1 – 6 are No. 6, 9, 12, 14, 15, and 16 in this trace.

Segment 1 sequence number: 1
 Segment 2 sequence number: 566
 Segment 3 sequence number: 2026
 Segment 4 sequence number: 3486

Segment 5 sequence number: 4946

Segment 6 sequence number: 6406

The sending time and the received time of ACKs are tabulated in the following table.

| | Sent time | ACK received time | RTT (seconds) |
|-----------|-----------|-------------------|---------------|
| Segment 1 | 0.026477 | 0.053937 | 0.02746 |
| Segment 2 | 0.041737 | 0.077294 | 0.035557 |
| Segment 3 | 0.054026 | 0.124085 | 0.070059 |
| Segment 4 | 0.054690 | 0.169118 | 0.11443 |
| Segment 5 | 0.077405 | 0.217299 | 0.13989 |
| Segment 6 | 0.078157 | 0.267802 | 0.18964 |

$$\text{EstimatedRTT} = 0.875 * \text{EstimatedRTT} + 0.125 * \text{SampleRTT}$$

EstimatedRTT after the receipt of the ACK of segment 1:

$$\text{EstimatedRTT} = \text{RTT for Segment 1} = 0.02746 \text{ second}$$

EstimatedRTT after the receipt of the ACK of segment 2:

$$\text{EstimatedRTT} = 0.875 * 0.02746 + 0.125 * 0.035557 = 0.0285$$

EstimatedRTT after the receipt of the ACK of segment 3:

$$\text{EstimatedRTT} = 0.875 * 0.0285 + 0.125 * 0.070059 = 0.0337$$

EstimatedRTT after the receipt of the ACK of segment 4:

$$\text{EstimatedRTT} = 0.875 * 0.0337 + 0.125 * 0.11443 = 0.0438$$

EstimatedRTT after the receipt of the ACK of segment 5:

$$\text{EstimatedRTT} = 0.875 * 0.0438 + 0.125 * 0.13989 = 0.0558$$

EstimatedRTT after the receipt of the ACK of segment 6:

$$\text{EstimatedRTT} = 0.875 * 0.0558 + 0.125 * 0.18964 = 0.0725 \text{ second}$$

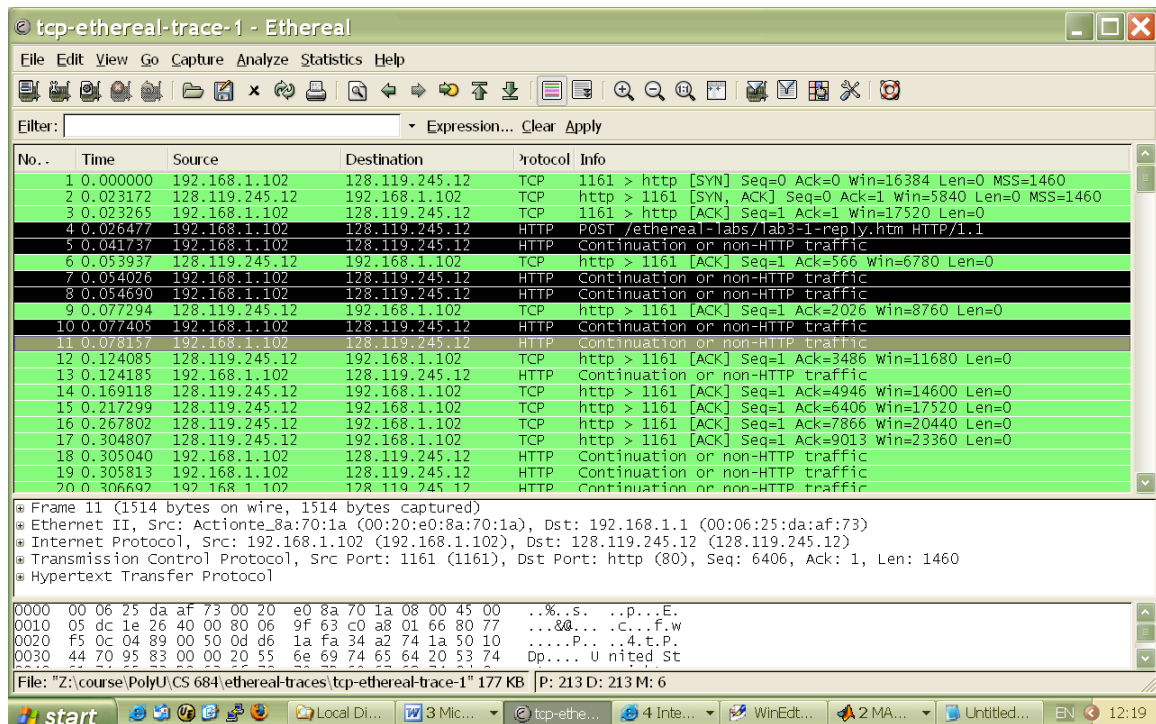


Figure 5: Segments 1 – 6

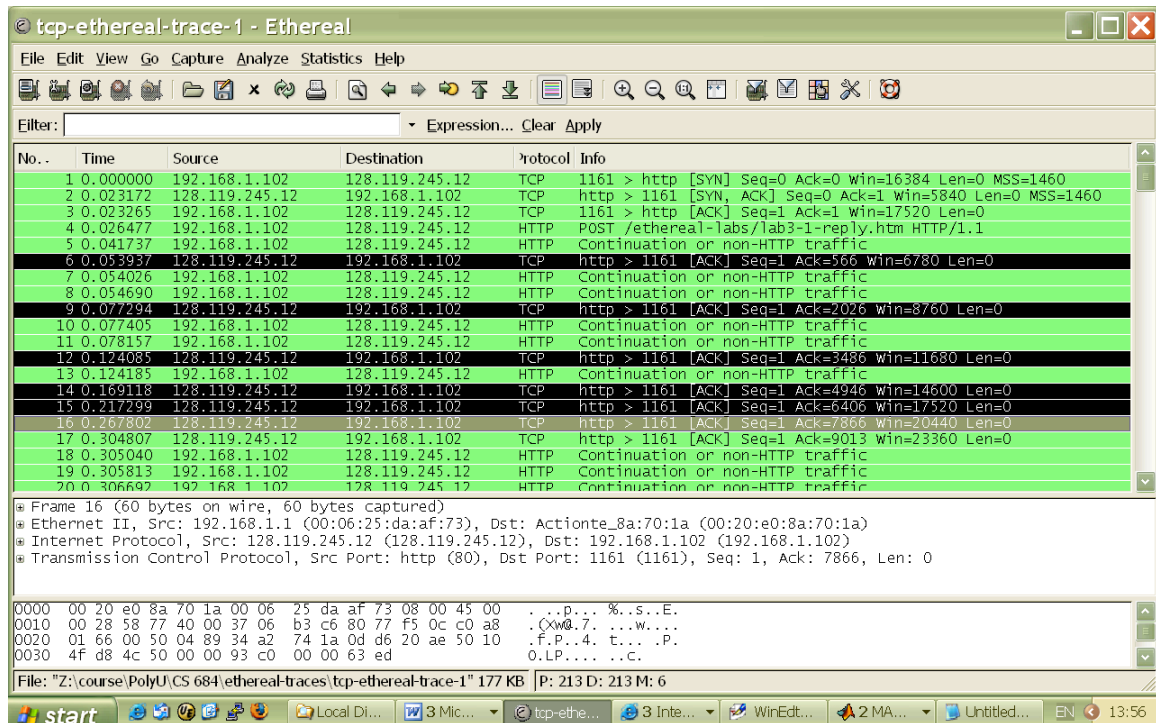


Figure 6: ACKs of segments 1 - 6

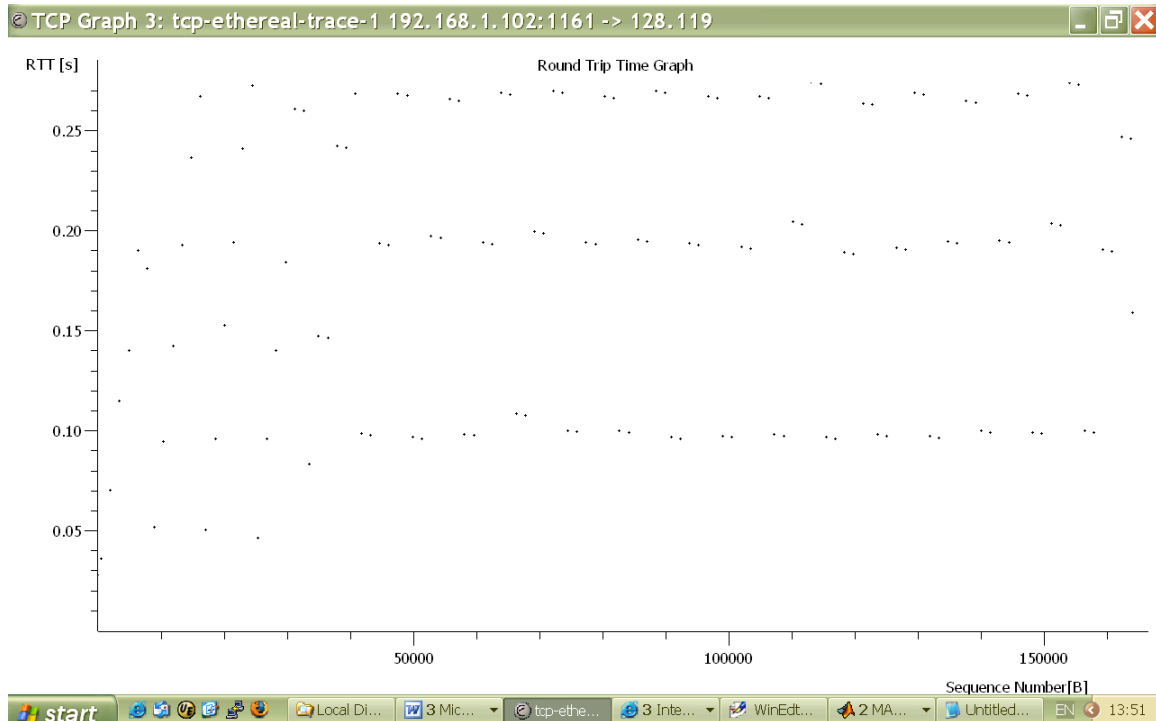


Figure 7: Round Trip Time Graph

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

Solution: Length of the first TCP segment (containing the HTTP POST): 565 bytes

Length of each of the other five TCP segments: 1460 bytes (MSS)

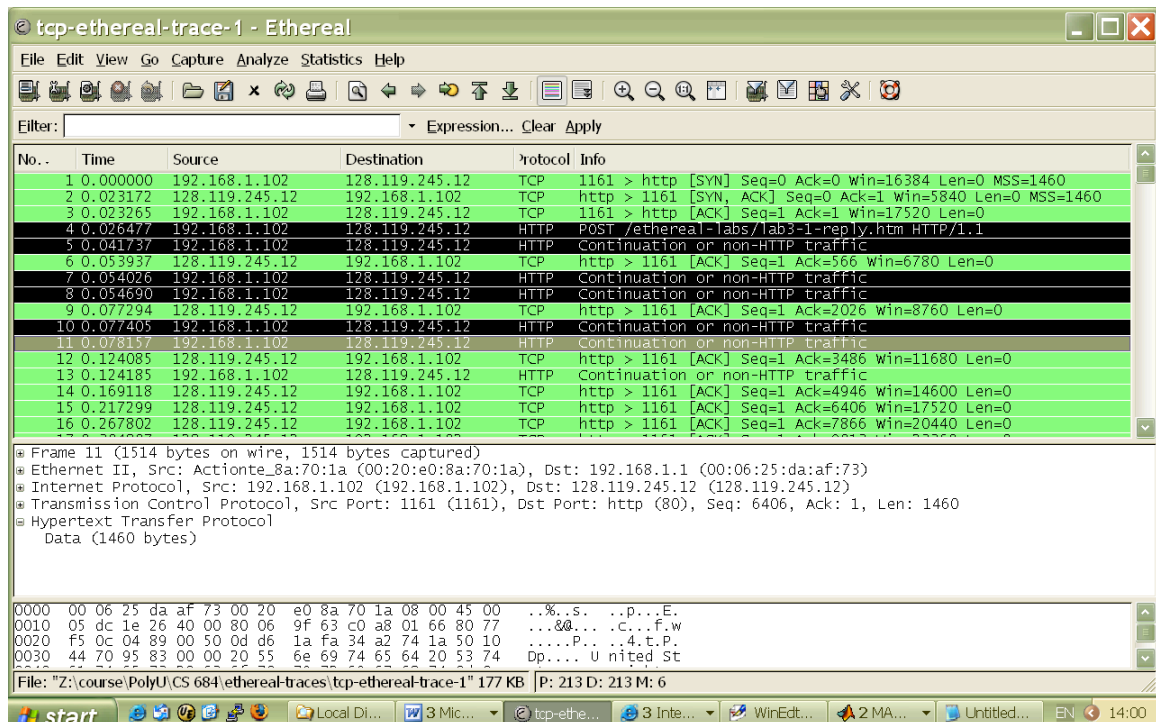


Figure 8: Lengths of segments 1 - 6

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?

Solution: The minimum amount of buffer space (receiver window) advertised at gaia.cs.umass.edu for the entire trace is 5840 bytes, which shows in the first acknowledgement from the server. This receiver window grows steadily until a maximum receiver buffer size of 62780 bytes. The sender is never throttled due to lacking of receiver buffer space by inspecting this trace.

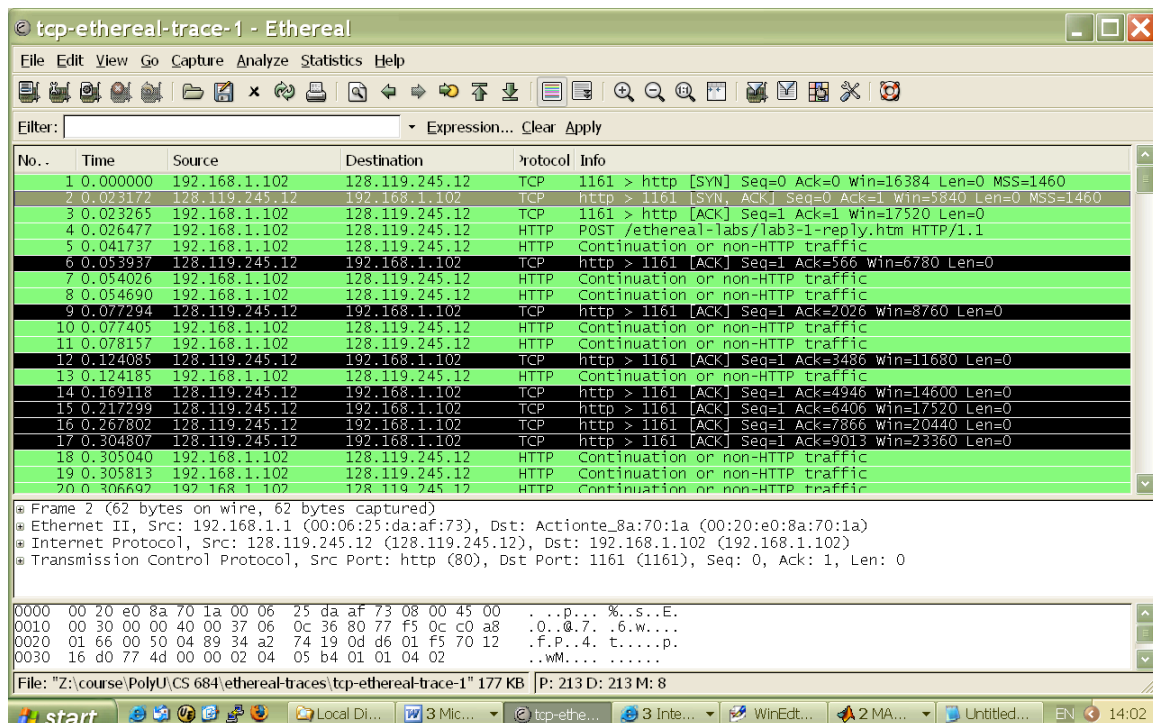


Figure 9: Minimum receive window advertised at gaia.cs.umass.edu (packet No. 2)

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

Solution: There are no retransmitted segments in the trace file. We can verify this by checking the sequence numbers of the TCP segments in the trace file. In the *Time-Sequence-Graph (Stevens)* of this trace, all sequence numbers from the source (192.168.1.102) to the destination (128.119.245.12) are increasing monotonically with respect to time. If there is a retransmitted segment, the sequence number of this retransmitted segment should be smaller than those of its neighboring segments.

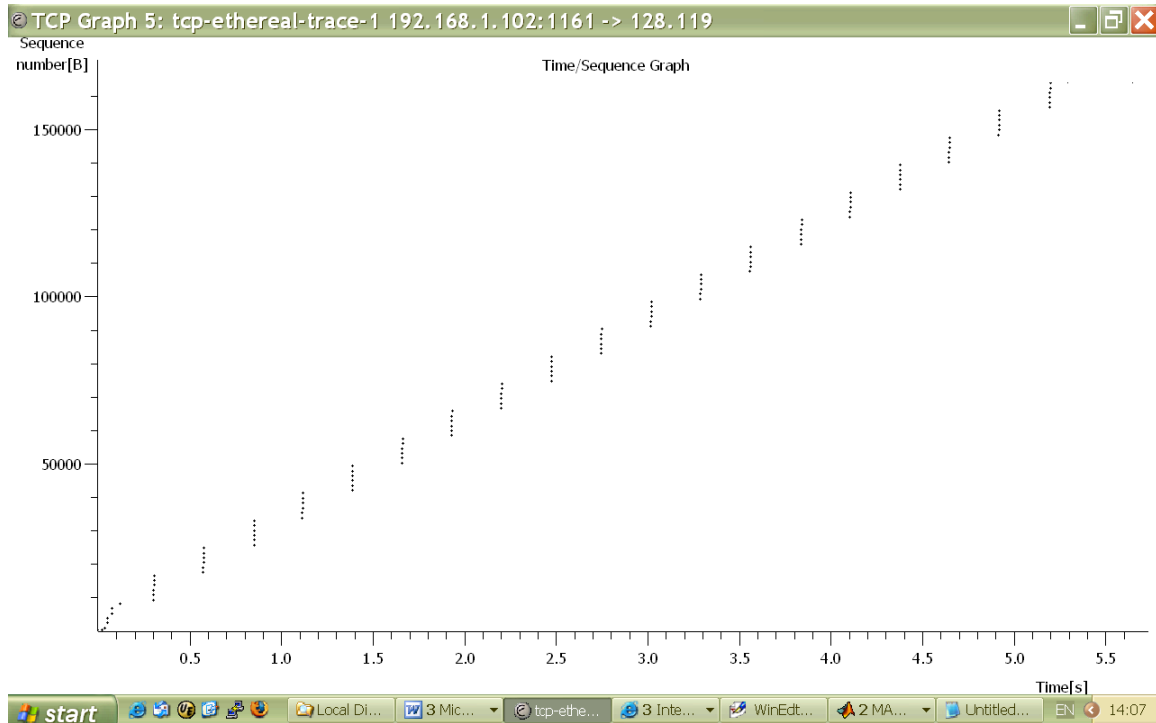


Figure 10: Sequence numbers of the segments from the source (192 . 168 . 1 . 102) to the destination (128 . 119 . 245 . 12)

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).

Solution: The acknowledged sequence numbers of the ACKs are listed as follows.

| | acknowledged sequence number | acknowledged data |
|--------|------------------------------|-------------------|
| ACK 1 | 566 | 566 |
| ACK 2 | 2026 | 1460 |
| ACK 3 | 3486 | 1460 |
| ACK 4 | 4946 | 1460 |
| ACK 5 | 6406 | 1460 |
| ACK 6 | 7866 | 1460 |
| ACK 7 | 9013 | 1147 |
| ACK 8 | 10473 | 1460 |
| ACK 9 | 11933 | 1460 |
| ACK 10 | 13393 | 1460 |
| ACK 11 | 14853 | 1460 |
| ACK 12 | 16313 | 1460 |

...

The difference between the acknowledged sequence numbers of two consecutive ACKs indicates the data received by the server between these two ACKs. By inspecting the amount of acknowledged data by each ACK, there are cases where the receiver is

ACKing every other segment. For example, segment of No. 80 acknowledged data with 2920 bytes = 1460*2 bytes.

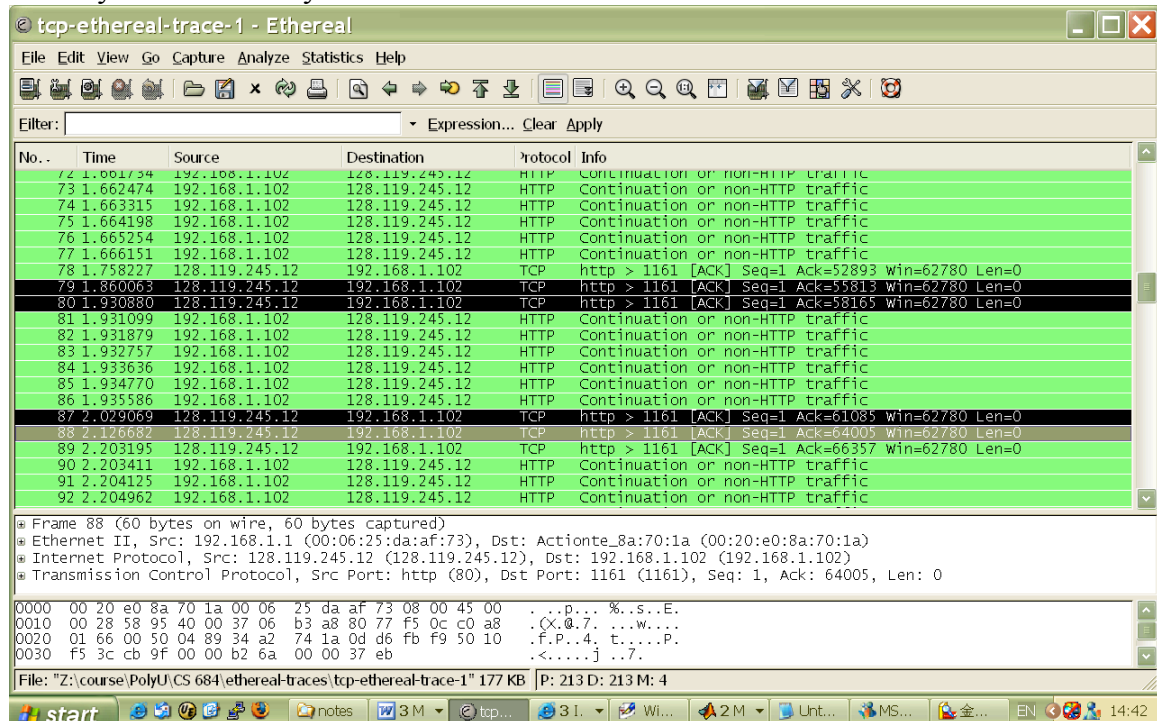


Figure 8: Cumulative ACKs (No. 80, 87, 88, etc) where the receiver is ACKing every other received segment.

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

Solution: The computation of TCP throughput largely depends on the selection of averaging time period. As a common throughput computation, in this question, we select the average time period as the whole connection time. Then, the average throughput for this TCP connection is computed as the ratio between the total amount 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 segment (i.e. 1 byte for No. 4 segment) and the acknowledged sequence number of the last ACK (164091 bytes for No. 202 segment). Therefore, the total data are $164091 - 1 = 164090$ bytes. The whole transmission time is the difference of the time instant of the first TCP segment (i.e., 0.026477 second for No.4 segment) and the time instant of the last ACK (i.e., 5.455830 second for No. 202 segment). Therefore, the total transmission time is $5.455830 - 0.026477 = 5.4294$ seconds. Hence, the throughput for the TCP connection is computed as $164090/5.4294 = 30.222$ KByte/sec.

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? Note that in this "real-world" trace, not everything is quite as neat and clean as in Figure 3.54 (also

note that the y-axis labels for the Time-Sequence-Graph(Stevens) plotting tool and Figure 3.54 are different).

Solution: TCP Slow Start begins at the start of the connection, i.e., when the HTTP POST segment is sent out. The identification of the TCP slow start phase and congestion avoidance phase depends on the value of the congestion window size of this TCP sender. However, the value of the congestion window size cannot be obtained directly from the Time-Sequence-Graph (Stevens) graph. Nevertheless, we can estimate the lower bound of the TCP window size by the amount of outstanding data because the outstanding data is the amount of data without acknowledgement. We also know that TCP window is constrained by the receiver window size and the receiver buffer can act as the upper bound of the TCP window size. In this trace, the receiver buffer is not the bottleneck; therefore, this upper bound is not quite useful to infer the TCP window size. Hence, we focus on the lower bound of the TCP window size.

From the following table, we cannot see that the amount outstanding data increases quickly at the start of this TCP flow; however, it never exceeds 8192 Bytes. Therefore, we can ensure that the TCP window size is larger than 8192 Bytes. Nevertheless, we cannot determine the end of the slow start phase and the start of the congestion avoidance phase for this trace. The major reason is that this TCP sender is not sending data aggressively enough to push to the congestion state. By inspecting the amount of outstanding data, we can observe that the application at most sends out a data block of 8192 bytes. Before it receives the acknowledgement for the whole block of these 8192 bytes, the application will not send more data. It indicates before the end of the slow start phase, the application already stops transmission temporally.

| Type | No. | Seq. | ACKed seq. | Outstanding data |
|------|-----|-------|------------|------------------|
| Data | 4 | 1 | | 565 |
| Data | 5 | 566 | | 2025 |
| ACK | 6 | | 566 | 1460 |
| Data | 7 | 2026 | | 2920 |
| Data | 8 | 3486 | | 4380 |
| ACK | 9 | | 2026 | 2920 |
| Data | 10 | 4946 | | 4380 |
| Data | 11 | 6406 | | 5840 |
| ACK | 12 | | 3486 | 4380 |
| Data | 13 | 7866 | | 5527 |
| ACK | 14 | | 4096 | 4917 |
| ACK | 15 | | 6006 | 3007 |
| ACK | 16 | | 7866 | 1147 |
| ACK | 17 | | 9013 | 0 |
| Data | 18 | 9013 | | 1460 |
| Data | 19 | 10473 | | 2920 |
| Data | 20 | 11933 | | 4380 |
| Data | 21 | 13393 | | 5840 |
| Data | 22 | 14853 | | 7300 |

| | | | | |
|------|----|-------|-------|------|
| Data | 23 | 16313 | | 8192 |
| ACK | 24 | | 10473 | 6732 |
| ACK | 25 | | 11933 | 5272 |
| ACK | 26 | | 13393 | 3812 |
| ACK | 27 | | 14853 | 2352 |
| ACK | 28 | | 16313 | 892 |
| ACK | 29 | | 17205 | 0 |
| Data | 30 | 17205 | | 1460 |
| Data | 31 | 18665 | | 2920 |
| Data | 32 | 20125 | | 4380 |
| Data | 33 | 21585 | | 5840 |
| Data | 34 | 23045 | | 7300 |
| Data | 35 | 24505 | | 8192 |
| ACK | 36 | | 18665 | 6732 |
| ACK | 37 | | 20125 | 5272 |
| ACK | 38 | | 21585 | 3812 |
| ACK | 39 | | 23045 | 2352 |
| ACK | 40 | | 24505 | 892 |
| ACK | 41 | | 25397 | 0 |
| Data | 42 | 25397 | | 1460 |
| Data | 43 | 26857 | | 2920 |
| Data | 44 | 28317 | | 4380 |
| Data | 45 | 29777 | | 5840 |
| Data | 46 | 31237 | | 7300 |
| Data | 47 | 32697 | | 8192 |
| ACK | 48 | | 26857 | |
| ACK | 49 | | 28317 | |
| ACK | 50 | | 29777 | |
| ACK | 51 | | 31237 | |
| ACK | 52 | | 33589 | |
| Data | 53 | 33589 | | 6732 |
| Data | 54 | 35049 | | 5272 |
| Data | 55 | 36509 | | 3812 |
| Data | 56 | 37969 | | 2352 |
| Data | 57 | 39429 | | 892 |
| Data | 58 | 40889 | | 0 |
| ACK | 59 | | 35049 | 6732 |
| ACK | 60 | | 37969 | 3812 |
| ACK | 61 | | 40889 | 892 |
| ACK | 62 | | 41781 | 0 |
| Data | 63 | 41781 | | 1460 |
| Data | 64 | 43241 | | 2920 |
| Data | 65 | 44701 | | 4380 |
| Data | 66 | 46161 | | 5840 |

| | | | | |
|------|----|-------|-------|------|
| Data | 67 | 47621 | | 7300 |
| Data | 68 | 49081 | | 8192 |
| ACK | 69 | | 44701 | 5272 |
| ACK | 70 | | 47621 | 2352 |
| ACK | 71 | | 49973 | 0 |
| Data | 72 | 49973 | | 1460 |
| Data | 73 | 51433 | | 2920 |
| Data | 74 | 52893 | | 4380 |
| Data | 75 | 54353 | | 5840 |
| Data | 76 | 55813 | | 7300 |
| Data | 77 | 57273 | | 8192 |
| ACK | 78 | | 52893 | 5272 |
| ACK | 79 | | 55813 | 2352 |
| ACK | 80 | | 58165 | 0 |
| Data | 81 | 58165 | | |

Note that the criteria to determine the end of slow start and the beginning of the congestion avoidance is the way how congestion window size reacts to the arrival of ACKs. Upon an ACK arrival, if the congestion window size increases by one MSS, TCP sender still stays in the slow start phase. In the congestion avoidance phase, the congestion window size increases at $1/(\text{current_congestion_window_size})$. By inspecting the change of the congestion window upon the arrival of ACKs, we can infer the states of the TCP sender.

Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.

Solution: The idealized behavior of TCP in the text assumes that TCP senders are aggressive in sending data. Too much traffic may congest the network; therefore, TCP senders should follow the AIMD algorithm so that when they detect network congestion (i.e., packet loss), their sending window size should drop down. In the practice, TCP behavior also largely depends on the application. In this example, when the TCP sender can send out data, there are no data available for transmission. In the web application, some of web objects have very small sizes. Before the end of slow start phase, the transmission is over; hence, the transmission of these small web objects suffers from the unnecessary long delay because of the slow start phase of TCP.