**Comp3331 lab04**

# Exercise 1: Understanding TCP using Wireshark

For this particular experiment download the trace file: tcp-wireshark-trace-1 .

The following indicate the steps for this experiment:

Step 1: Start Wireshark by typing *wireshark* at the command prompt.

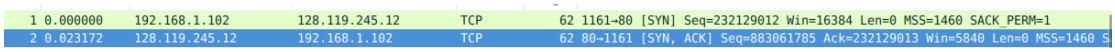
Step 2: Load the trace file *tcp-ethereal-trace-1* by using the *File* pull down menu, choosing *Open* and selecting the appropriate trace file. This file captures the sequence of messages exchanged between a host and a remote server (gaia.cs.umass.edu). The host transfers a 150 KB text file, which contains the text of Lewis Carrol's *Alice's Adventure in Wonderland* to the server. Note that the file is being transferred from the host to the server using a HTTP POST message.

Step 3: Now filter out all non-TCP packets by typing "tcp" (without quotes) in the filter field towards the top of the Wireshark window. You should see a series of TCP segments between the host in MIT and gaia.cs.umass.edu. The first three segments of the trace consist of the initial three-way handshake containing the SYN, SYN ACK and ACK messages. You should see an HTTP POST message in the 4 th segment of the trace being sent from the host in MIT to gaia.cs.umass.edu (check the contents of the payload of this segment). You should observe that the text file is transmitted as multiple TCP segments (i.e. a single POST message has been split into several TCP segments) from the client to the server (gaia.cs.umass.edu). You should also see several TCP ACK segments been returned in the reverse direction.

**IMPORTANT NOTE:** Do the sequence numbers for the sender and receiver start from zero? The reason for this is that Wireshark by default scales down all real sequence numbers such that the first segment in the trace file always starts from 0. To turn off this feature, you have to click Edit->Preferences>Protocols->TCP (or Wireshark->Preferences->Protocols->TCP) and then disable the "Relative Sequence Numbers" option. Note that the answers in the solution set will reflect this change. If you conduct the experiment without this change, the sequence numbers that you observe will be different from the ones in the answers. Also, set the time shown in the 2nd column as the "Seconds since beginning of capture" under view->Time display format.

*Question 1* . What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection? 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?

**Answer:**

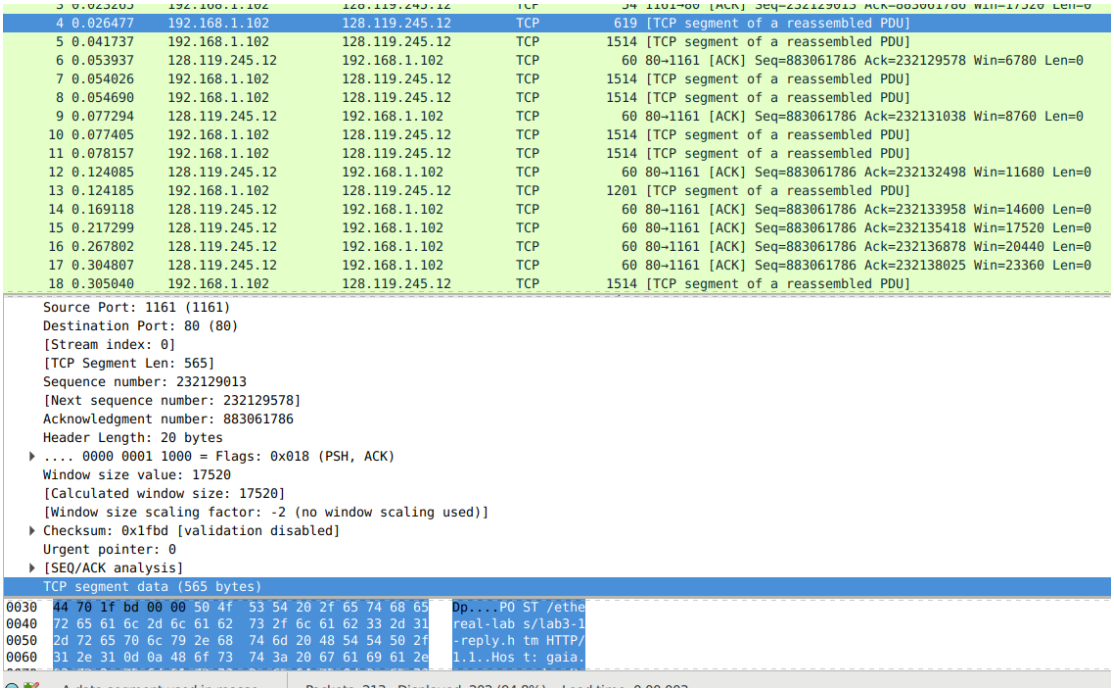



IP of gaia.cs.umass.edu : 128.119.245.12 Using port 80

IP of host: 192.168.1.102 Port 1161

*Question 2.* 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.

**Answer:**

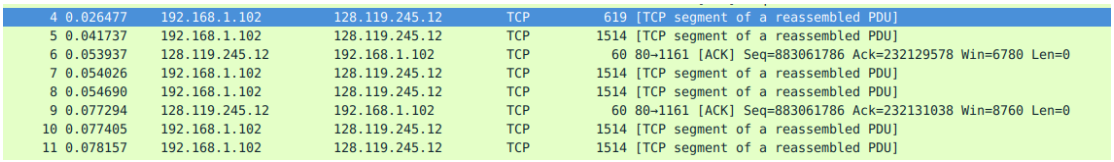



the sequence number is **232129013.**

*Question 3.* 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) sent from the client to the web server (Do not consider the ACKs received from the server as part of these six segments)? 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 relevant parts of Section 3.5 or lecture slides) after the receipt of each ACK? Assume that the initial value of *EstimatedRTT* is equal to the measured RTT ( *SampleRTT* ) for the first segment, and then is computed using the *EstimatedRTT* equation for all subsequent segments. Set alpha to 0.125.

**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* . However, do not use this graph to answer the above question.

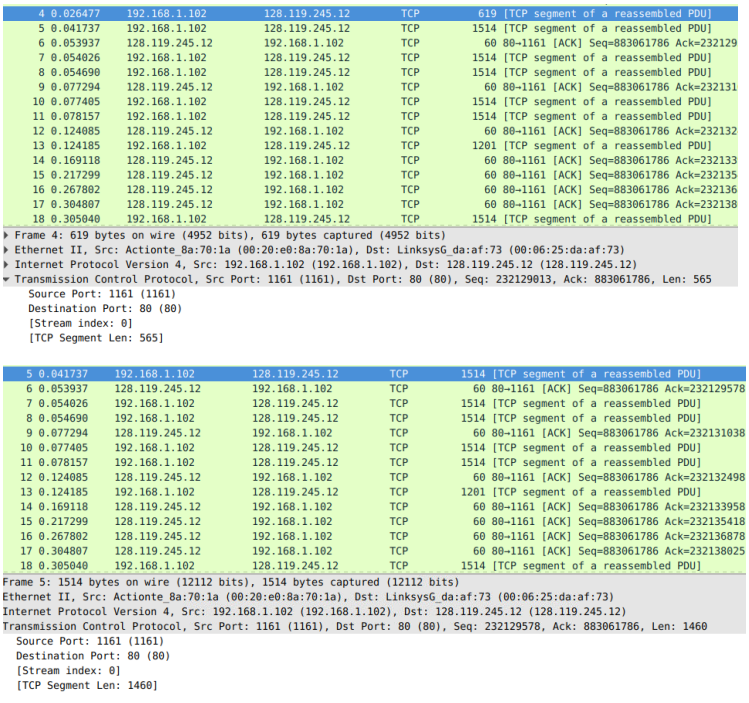
**Answer:**

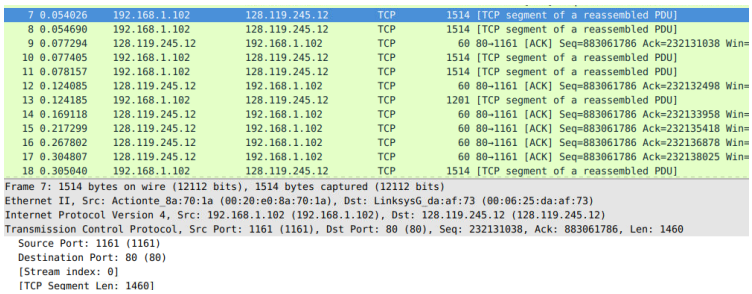


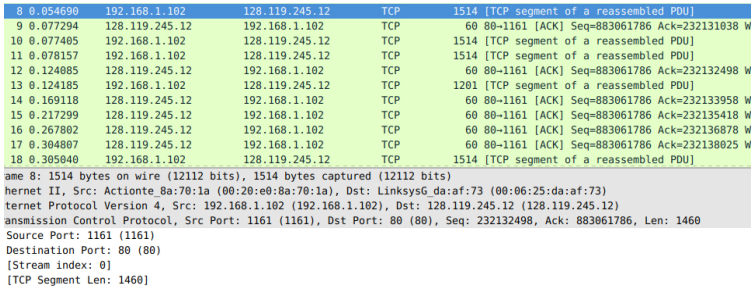

| No. | sequence | Time sent(s) | Ack receive(s) | RTT(diff) | *EstimatedRTT* |
|---|---|---|---|---|---|
| 1(4) | 232129013 | 0.026477 | 0.053937 | 0.02746 | 0.02746 |
| 2(5) | 232129578 | 0.041737 | 0.077294 | 0.035557 | 0.028472 |
| 3(7) | 232131038 | 0.054026 | 0.124085 | 0.070059 | 0.03367 |
| 4(8) | 232132498 | 0.054690 | 0.169118 | 0.114428 | 0.043765 |
| 5(10) | 232133958 | 0.077405 | 0.217299 | 0.139894 | 0.055781 |
| 6(11) | 232135418 | 0.078157 | 0.267802 | 0.189645 | 0.072514 |

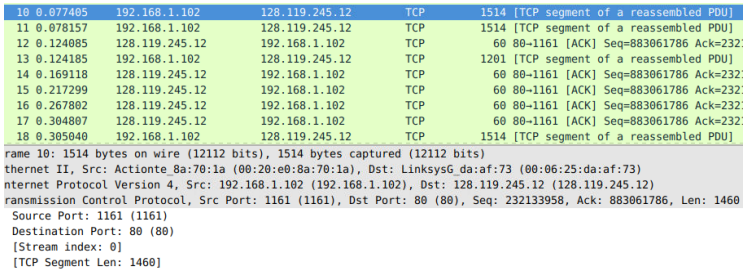
*Question 4.* What is the length of each of the first six TCP segments?

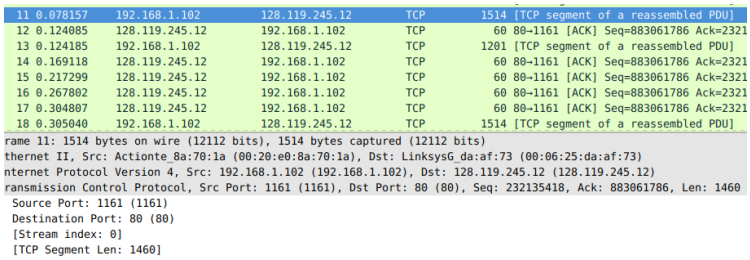
**Answer:**





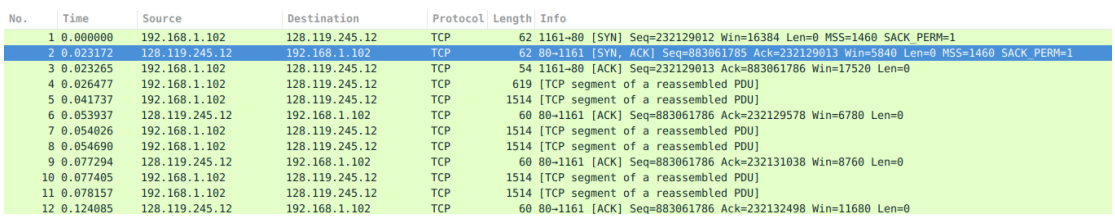




565,1460,1460,1460,1460,1460

*Question 5.* 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?

**Answer:**



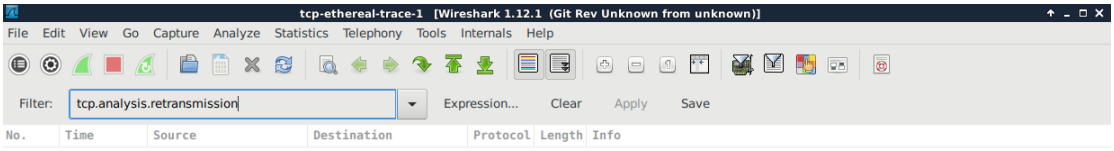

5840

No, it does not. Since the windows is growing.

Question 6. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

**Answer:**

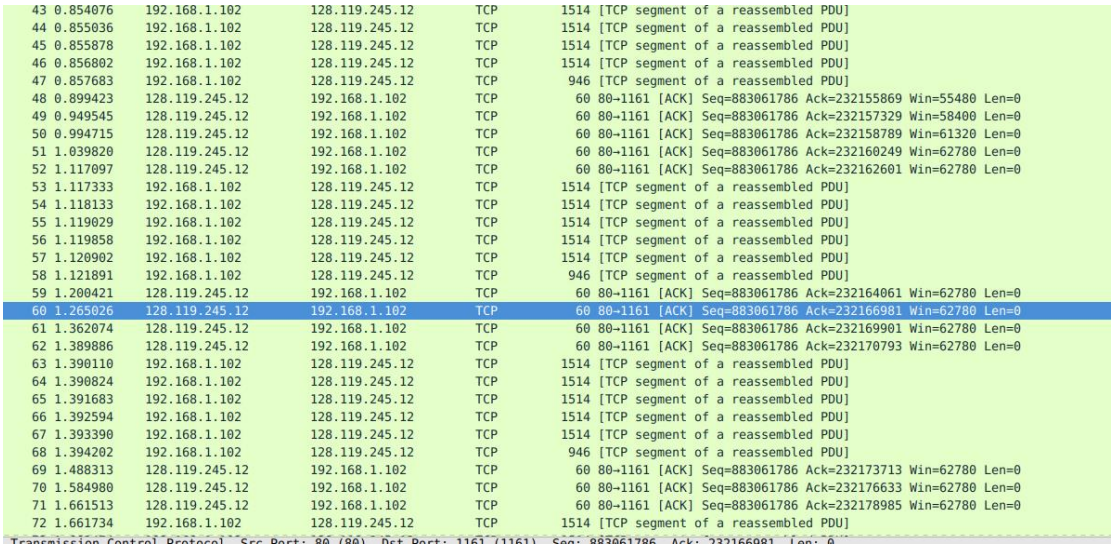
No, there is no retransmitted segments in the trace file. tcp.analysis.retransmission in the filter.




*Question 7.* How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (recall the discussion about delayed acks from the lecture notes or Section 3.5 of the text).

**Answer:**

It is 1460 bytes.



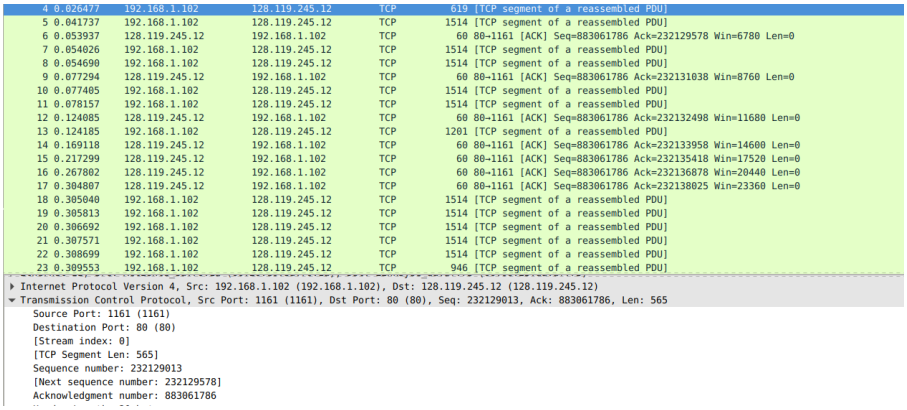
| | | | | | |
|---|---|---|---|---|---|
| 43 0.854076 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 44 0.855036 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 45 0.855878 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 46 0.856802 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 47 0.857683 | 192.168.1.102 | 128.119.245.12 | TCP | 946 | [TCP segment of a reassembled PDU] |
| 48 0.899423 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232155869 Win=55480 Len=0 |
| 49 0.949545 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232157329 Win=58400 Len=0 |
| 50 0.994715 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232158789 Win=61320 Len=0 |
| 51 1.039820 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232160249 Win=62780 Len=0 |
| 52 1.117097 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232162601 Win=62780 Len=0 |
| 53 1.117333 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 54 1.118133 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 55 1.119029 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 56 1.119858 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 57 1.120902 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 58 1.121891 | 192.168.1.102 | 128.119.245.12 | TCP | 946 | [TCP segment of a reassembled PDU] |
| 59 1.200421 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232164061 Win=62780 Len=0 |
| 60 1.265026 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232166981 Win=62780 Len=0 |
| 61 1.362074 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232169901 Win=62780 Len=0 |
| 62 1.389886 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232170793 Win=62780 Len=0 |
| 63 1.390110 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 64 1.390824 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 65 1.391683 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 66 1.392594 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 67 1.393390 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 68 1.394202 | 192.168.1.102 | 128.119.245.12 | TCP | 946 | [TCP segment of a reassembled PDU] |
| 69 1.488313 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232173713 Win=62780 Len=0 |
| 70 1.584980 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232176633 Win=62780 Len=0 |
| 71 1.661513 | 128.119.245.12 | 192.168.1.102 | TCP | 60 | 80→1161 [ACK] Seq=883061786 Ack=232178985 Win=62780 Len=0 |
| 72 1.661734 | 192.168.1.102 | 128.119.245.12 | TCP | 1514 | [TCP segment of a reassembled PDU] |

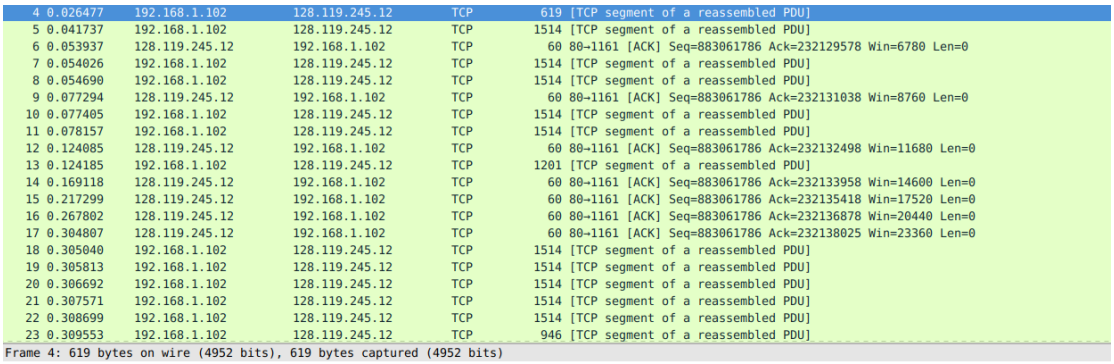
In the beginning, receiver will ack each packet. But at #60 , The ack 232166981 is ack two segment with seq num 232164061 and seq num 232165521. The receiver sends a cumulative ACK for two segments. And after # 60 there are many cases that receiver send ack for two segments received. This is due to TCP use delayed ACK wait up to 500ms for next segment and send cumulative ack for received segments. If there is no next segment, send ACK.

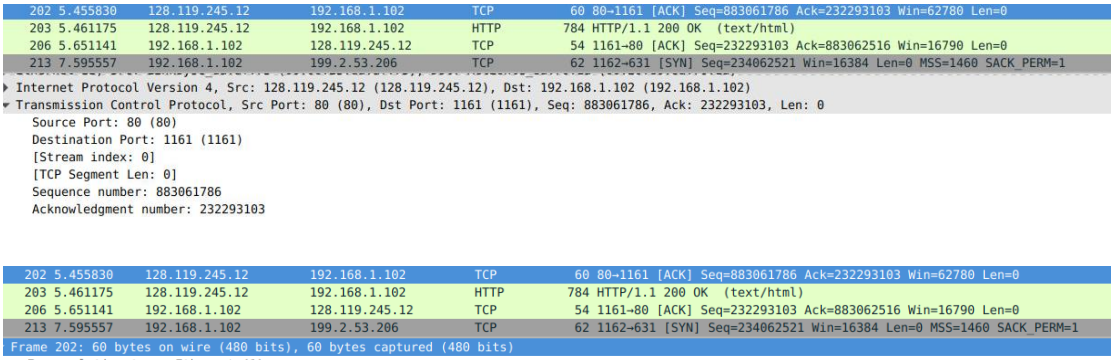
*Question 8.* What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

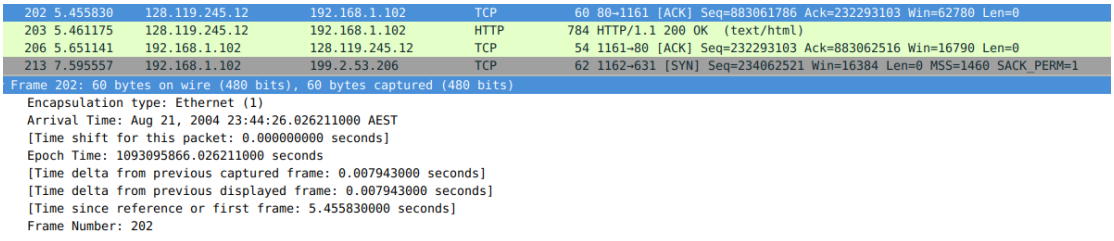
**Answer:**

Throughput = $\frac{Total\ amout\ data}{Total\ transmission\ time}$




Total amount data = #202 ack - #4 seq = 232293103 – 232129013 = 164090 bytes

Total transmission time = #202 ack time (5.455830) - #4 time (0.026477) = 5.429353 seconds

Throughput = $\frac{164090\ bytes}{5.429353\ seconds}$ = 30222.7539819 bytes/s = 30.223KByte/s

# Exercise 2: TCP Connection Management

Consider the following TCP transaction between a client (10.9.16.201) and a server (10.99.6.175).

| No | Source IP | Destination IP | Protocol | Info |
|---|---|---|---|---|
| 295 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [SYN] Seq=2818463618 win=8192 MSS=1460 |
| 296 | 10.99.6.175 | 10.9.16.201 | TCP | 5000 > 50045 [SYN, ACK] Seq=1247095790 Ack=2818463619 win=262144 MSS=1460 |
| 297 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [ACK] Seq=2818463619 Ack=1247095791 win=65535 |
| 298 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [PSH, ACK] Seq=2818463619 Ack=1247095791 win=65535 |
| 301 | 10.99.6.175 | 10.9.16.201 | TCP | 5000 > 50045 [ACK] Seq=1247095791 Ack=2818463652 win=262096 |
| 302 | 10.99.6.175 | 10.9.16.201 | TCP | 5000 > 50045 [PSH, ACK] Seq=1247095791 Ack=2818463652 win=262144 |
| 303 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [ACK] Seq=2818463652 Ack=1247095831 win=65535 |
| 304 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [FIN, ACK] Seq=2818463652 Ack=1247095831 win=65535 |
| 305 | 10.99.6.175 | 10.9.16.201 | TCP | 5000 > 50045 [FIN, ACK] Seq=1247095831 Ack=2818463652 win=262144 |
| 306 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [ACK] Seq=2818463652 Ack=1247095832 win=65535 |
| 308 | 10.99.6.175 | 10.9.16.201 | TCP | 5000 > 50045 [ACK] Seq=1247095831 Ack=2818463653 win=262144 |

Answer the following questions:

*Question 1* . What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and server?

**Answer:**

281843618

*Question 2.* What is the sequence number of the SYNACK segment sent by the server to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did the server determine that value?

**Answer:**

Sequence number :1247095790 Ack:2818463619

Determined by sequence number from client + 1.

*Question 3* . What is the sequence number of the ACK segment sent by the client computer in response to the SYNACK? What is the value of the Acknowledgment field in this ACK segment? Does this segment contain any data?

**Answer:**

Sequence number :2818463619 Ack:1247095791 No, it does not contain any data. Since the No.298 sequence number is same.

*Question 4* . Who has done the active close? client or the server? how you have determined this? What type of closure has been performed? 3 Segment (FIN/FINACK/ACK), 4 Segment (FIN/ACK/FIN/ACK) or Simultaneous close?

**Answer:**

| | | | | |
|---|---|---|---|---|
| 304 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [FIN, ACK] Seq=2818463652 Ack=1247095831 win=65535 |
| 305 | 10.99.6.175 | 10.9.16.201 | TCP | 5000 > 50045 [FIN, ACK] Seq=1247095831 Ack=2818463652 win=262144 |
| 306 | 10.9.16.201 | 10.99.6.175 | TCP | 50045 > 5000 [ACK] Seq=2818463652 Ack=1247095832 win=65535 |
| 308 | 10.99.6.175 | 10.9.16.201 | TCP | 5000 > 50045 [ACK] Seq=1247095831 Ack=2818463653 win=262144 |

Both client and server have done the active close. Since both client and server sent [FIN,ACK] to each other without received FIN flag. And the 304 sequence number is 305 ack, instead of seq + 1.After that both send ack to each other. And ack is equal the sequence num + 1,which means they both get fin flag. Then close the connection.

The type is simultaneous close.

*Question 5* . How many data bytes have been transferred from the client to the server and from the server to the client during the whole duration of the connection? What relationship does this have with the Initial Sequence Number and the final ACK received from the other side?

**Answer:**

From client to server: 2818463653 – 2818463618 – 2(SYN, FIN) = 33 Bytes

From server to client:1247095832 – 1247095790 – 2(SYN,FIN) = 40 Bytes

Since the segment with Sequence Number and data sent to other side, other side will expected that next segment contain the sequence number is previous number plus previous data amount, which is ack num other sider sent to, which can use calculate the data transferred.

The SYN, FIN segment will increase 1 for ack num, but contain nothing data.