Exercise 1: Understanding TCP using Wireshark

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

The following indicate the steps for this experiment:

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

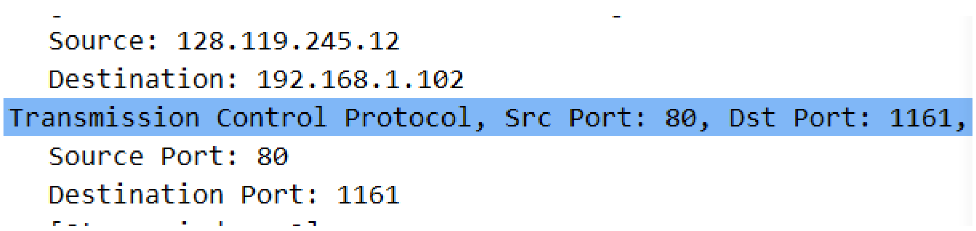
Step 2: Load the trace file *tcp-ethereal-trace-1* by using the *File* pull down menu, choosing *Open* and selecting the appropriate trace file. This file captures the sequence of messages exchanged between a host and a remote server (gaia.cs.umass.edu). The host transfers a 150 KB text file, which contains 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.

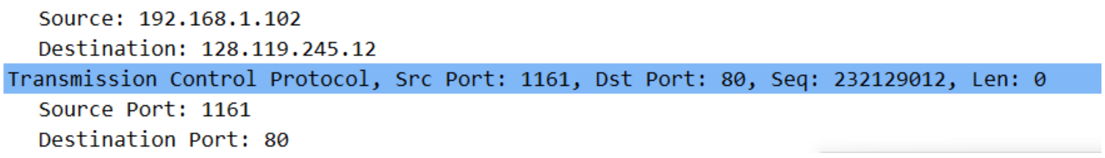
IMPORTANT NOTE: Do the sequence numbers for the sender and receiver start from zero? The reason for this is that Wireshark by default scales down all real sequence numbers such that the first segment in the trace file always starts from 0. To turn off this feature, you have to click Edit->Preferences>Protocols->TCP (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: The IP address of gaia.cs.umass.edu is 128.119.245.12. And the sending and receiving segments port is 80. The IP address of client computer is 192.168.1.102, the source port is 1161.



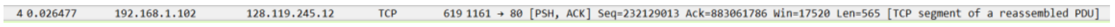
```
    Source: 128.119.245.12
    Destination: 192.168.1.102
Transmission Control Protocol, Src Port: 80, Dst Port: 1161,
    Source Port: 80
    Destination Port: 1161
```



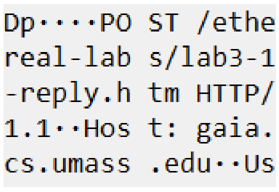
```
    Source: 192.168.1.102
    Destination: 128.119.245.12
Transmission Control Protocol, Src Port: 1161, Dst Port: 80, Seq: 232129012, Len: 0
    Source Port: 1161
    Destination Port: 80
```

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 Ethereal window, looking for a segment with a "POST" within its DATA field.

Answer: The sequence number is 232129013. This is because we can find the POST command in 4th TCP segment like below, so we can find the sequence number.



```
4 0.026477    192.168.1.102    128.119.245.12    TCP    619 1161 → 80 [PSH, ACK] Seq=232129013 Ack=883061786 Win=17520 Len=565 [TCP segment of a reassembled PDU]
```



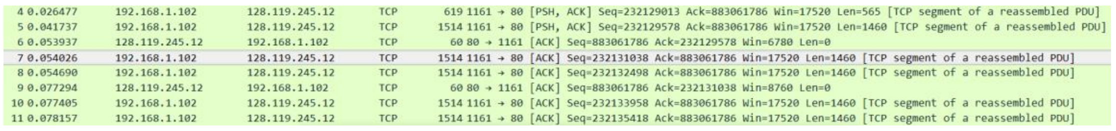
```
Dp····PO ST /ethe
real-lab s/lab3-1
-reply.h tm HTTP/
1.1··Hos t: gaia.
cs.umass .edu··Us
```

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.

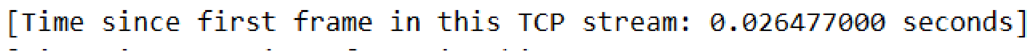
Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: *Statistics->TCP*

Stream Graph>Round Trip Time Graph . However, do not use this graph to answer the above question.

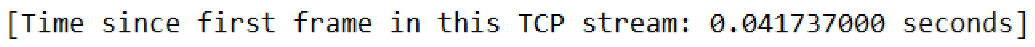
Answer: The sequence number is 232129013, 232129578, 232131038, 232132498, 232133958, 232135418 respectively. We can find that the sending time is 0.026477, 0.041737, 0.054026, 0.054690, 0.077405, 0.078157 respectively.



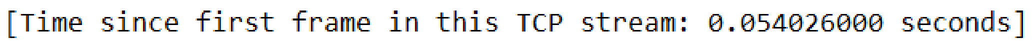
```
 4 0.026477   192.168.1.102   128.119.245.12   TCP    619 1161 → 80 [PSH, ACK] Seq=232129013 Ack=883061786 Win=17520 Len=565 [TCP segment of a reassembled PDU]
 5 0.041737   192.168.1.102   128.119.245.12   TCP   1514 1161 → 80 [PSH, ACK] Seq=232129578 Ack=883061786 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
 6 0.053937   128.119.245.12  192.168.1.102    TCP     60 80 → 1161 [ACK] Seq=883061786 Ack=232129578 Win=6780 Len=0
 7 0.054026   192.168.1.102   128.119.245.12   TCP   1514 1161 → 80 [ACK] Seq=232131038 Ack=883061786 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=232132498 Ack=883061786 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=883061786 Ack=232131038 Win=8760 Len=0
10 0.077405   192.168.1.102   128.119.245.12   TCP   1514 1161 → 80 [ACK] Seq=232133958 Ack=883061786 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=232135418 Ack=883061786 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
```



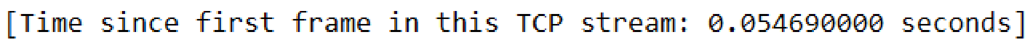
```
[Time since first frame in this TCP stream: 0.026477000 seconds]
```



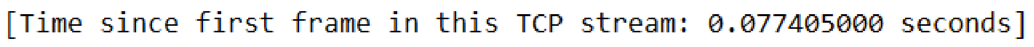
```
[Time since first frame in this TCP stream: 0.041737000 seconds]
```



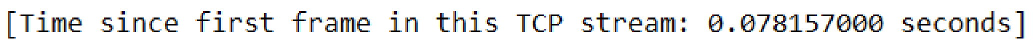
```
[Time since first frame in this TCP stream: 0.054026000 seconds]
```



```
[Time since first frame in this TCP stream: 0.054690000 seconds]
```

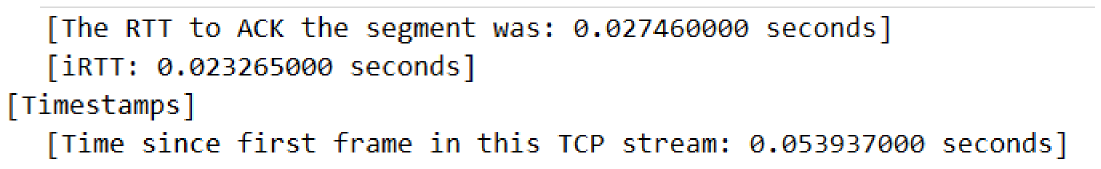


```
[Time since first frame in this TCP stream: 0.077405000 seconds]
```

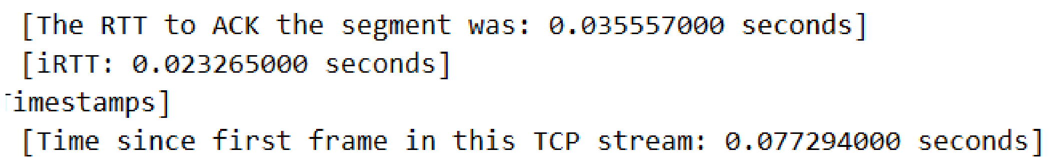


```
[Time since first frame in this TCP stream: 0.078157000 seconds]
```

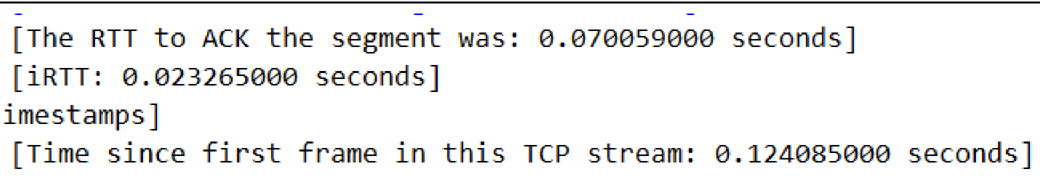
According to ACK segments for each segment, we can find that receiving time is 0.053937, 0.077294, 0.124085, 0.169118, 0.217299, 0.267802 respectively. And also, we can find the RTT is 0.02746, 0.035557, 0.070059, 0.114428, 0.139894, 0.189645 respectively.



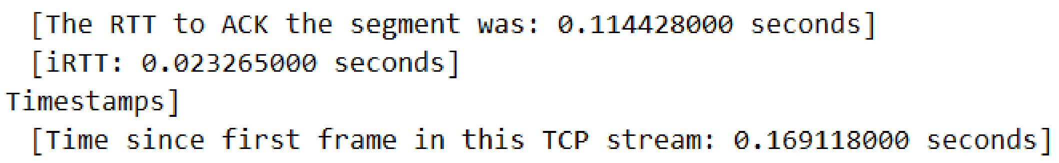
```
   [The RTT to ACK the segment was: 0.027460000 seconds]
   [iRTT: 0.023265000 seconds]
[Timestamps]
   [Time since first frame in this TCP stream: 0.053937000 seconds]
```



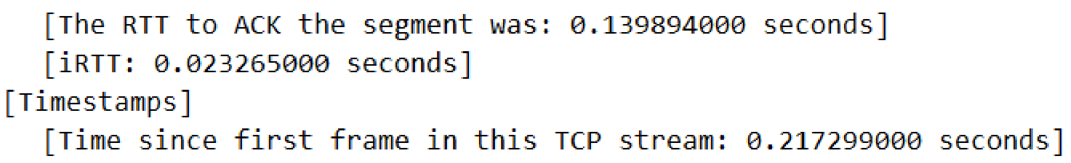
```
 [The RTT to ACK the segment was: 0.035557000 seconds]
 [iRTT: 0.023265000 seconds]
imestamps]
 [Time since first frame in this TCP stream: 0.077294000 seconds]
```



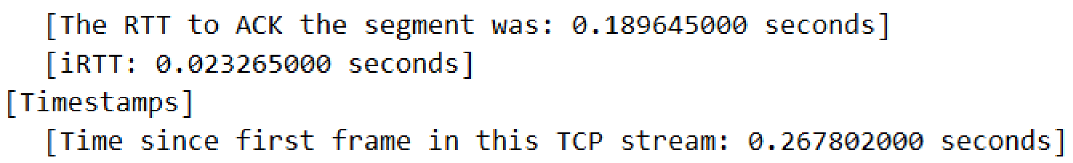
```
 [The RTT to ACK the segment was: 0.070059000 seconds]
 [iRTT: 0.023265000 seconds]
imestamps]
 [Time since first frame in this TCP stream: 0.124085000 seconds]
```



```
  [The RTT to ACK the segment was: 0.114428000 seconds]
  [iRTT: 0.023265000 seconds]
Timestamps]
  [Time since first frame in this TCP stream: 0.169118000 seconds]
```



```
   [The RTT to ACK the segment was: 0.139894000 seconds]
   [iRTT: 0.023265000 seconds]
[Timestamps]
   [Time since first frame in this TCP stream: 0.217299000 seconds]
```



```
   [The RTT to ACK the segment was: 0.189645000 seconds]
   [iRTT: 0.023265000 seconds]
[Timestamps]
   [Time since first frame in this TCP stream: 0.267802000 seconds]
```

EstimatedRTT = (1- α) EstimatedRTT + α SampleRTT

If α = 0.125, then we can get this formula:

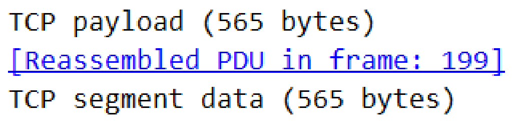
EstimatedRTT = 0.875 * EstimatedRTT + 0.125 * SampleRTT

Therefore, we can get EstimatedRTT for each segment:

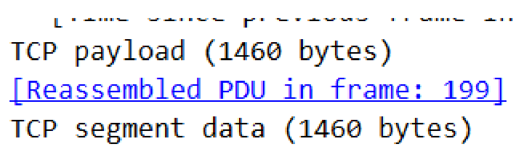
0.02746, 0.028472, 0.03367, 0.043765, 0.055781, 0.072514

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

Answer: The length of each of TCP segment is 565 bytes, 1460 bytes, 1460 bytes, 1460 bytes, 1460 bytes, 1460 bytes.



```
TCP payload (565 bytes)
[Reassembled PDU in frame: 199]
TCP segment data (565 bytes)
```

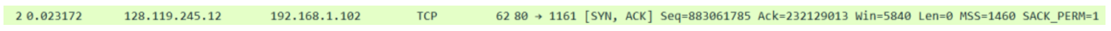


```
TCP payload (1460 bytes)
[Reassembled PDU in frame: 199]
TCP segment data (1460 bytes)
```

…

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: We can find that the minimum advertised window is 5840 through examining all the ACK. And this is advertised in the SYNACK segment. The receiver window doesn’t seem to throttle the sender. When the receiver window is the minimum value (5840), the sender doesn’t seem to be affected.



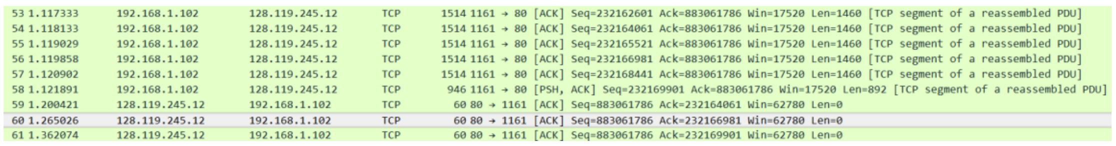
```
2 0.023172    128.119.245.12    192.168.1.102    TCP    62 80 → 1161 [SYN, ACK] Seq=883061785 Ack=232129013 Win=5840 Len=0 MSS=1460 SACK_PERM=1
```

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: There are no retransmitted segments in the trace file. If there are retransmitted segments in the trace file, there would be some same sequence numbers in the trace file. However, we can find that all sequence numbers from the source (192.168.1.102) to the destination (128.119.245.12) are increasing monotonically. If there are some retransmitted segments, the sequence numbers will have some same numbers but not increasing monotonically.

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: Generally, each packet is individually ACKed by the receiver, but when the sender transmits much more packets, especially in 60th packets, we can find that the 232166981 is ACKing the 232164061 and the 232165521. Therefore, we can know that when sender sends a large number of packets which may occur some packets loss or timeout, the receiver may use a cumulative ACK for two TCP segments that it receives. In fact, the TCP uses delayed acks to make receiver wait for up to 500msec for another arrival of next segment and then it sends a cumulative ACK for these two segments.



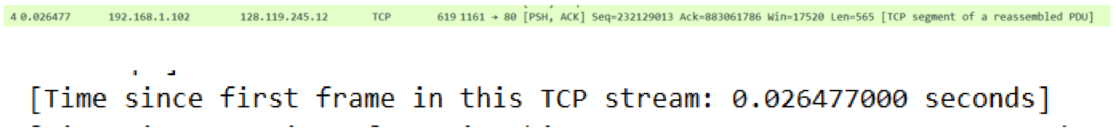
```
53 1.117333    192.168.1.102     128.119.245.12    TCP    1514 1161 → 80 [ACK] Seq=232162601 Ack=883061786 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
54 1.118133    192.168.1.102     128.119.245.12    TCP    1514 1161 → 80 [ACK] Seq=232164061 Ack=883061786 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
55 1.119029    192.168.1.102     128.119.245.12    TCP    1514 1161 → 80 [ACK] Seq=232165521 Ack=883061786 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
56 1.119858    192.168.1.102     128.119.245.12    TCP    1514 1161 → 80 [ACK] Seq=232166981 Ack=883061786 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
57 1.120902    192.168.1.102     128.119.245.12    TCP    1514 1161 → 80 [ACK] Seq=232168441 Ack=883061786 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
58 1.121891    192.168.1.102     128.119.245.12    TCP     946 1161 → 80 [PSH, ACK] Seq=232169901 Ack=883061786 Win=17520 Len=892 [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
```

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

Answer: Since Throughput = Transferred Bytes / Transmission Time, we just need to find the Bytes and the Time. Since the connection establishment phase of TCP, there are no data transferred. We can use the 4th segment (seq number: 232129013)

to the last segment that the receiver ACKed (seq number: 232293103) to calculate the bytes of data transferred. And also, the seq number is associated with data bytes, thus, Transferred Bytes = 232293103 – 232129013 = 164090 Bytes. We can also get the time from 4th segment to the last segment. As below, we can get the Transmission Time is 5.455830(the last segment) - 0.026477(the 4th segment) = 5.429353 sec. Therefore, the throughput for the TCP connection is 164090 / 5.429353, and the result is approximately equal to 30.222 Kbyte/sec.

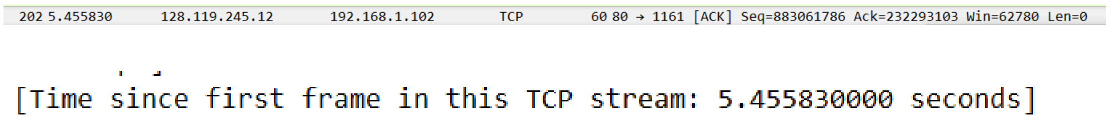
4th segment:



```
4 0.026477    192.168.1.102    128.119.245.12    TCP    619 1161 → 80 [PSH, ACK] Seq=232129013 Ack=883061786 Win=17520 Len=565 [TCP segment of a reassembled PDU]

[Time since first frame in this TCP stream: 0.026477000 seconds]
```

The last segment:

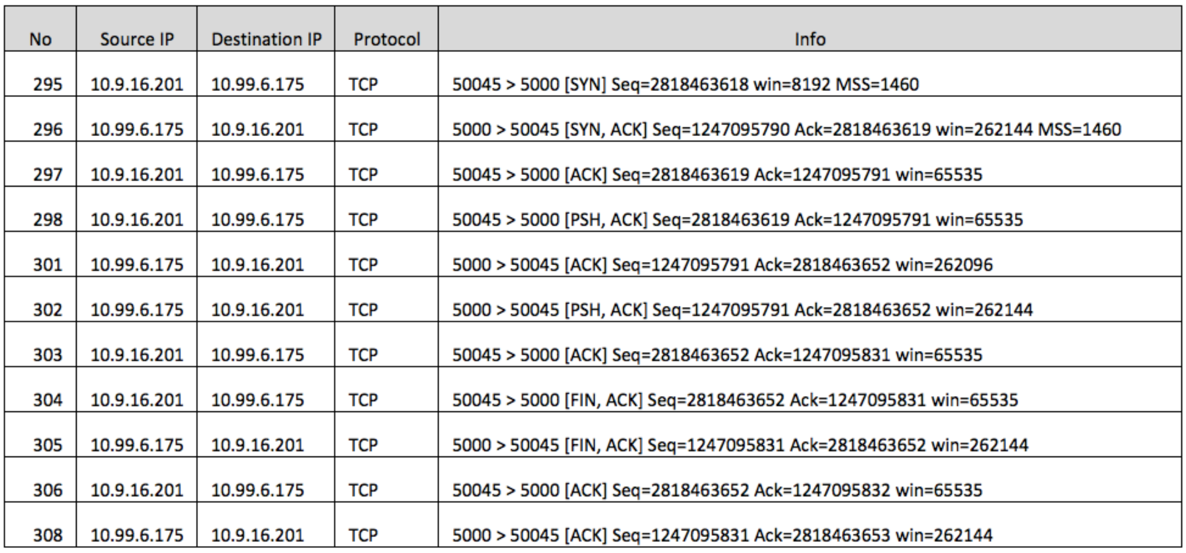


```
202 5.455830    128.119.245.12    192.168.1.102    TCP    60 80 → 1161 [ACK] Seq=883061786 Ack=232293103 Win=62780 Len=0

[Time since first frame in this TCP stream: 5.455830000 seconds]
```

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: The sequence number of the TCP SYN segment that is used to initiate the TCP connection from client is 2818463618 (No.295 segment). In the Flags field of TCP packets, the SYN flag is set to 1 which means it is a SYN packet.

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: The sequence number of the SYNACK segment sent by the server is 1247095790 (No. 296 segment). The value of Acknowledgement field is 2818463619. The SYNACK packet is acknowledging the SYN packet from client, and since the SYN packet from client doesn't contain any data, the server increases the sequence number (2818463618) from client by 1 and copies it to the Acknowledgement field. That means server wants to get the next segment from the client and the segment's sequence number is 2818463619. In the Flags field of TCP packets, the SYN flag and the ACK flag are both set to 1 which means it is a SYNACK packet.

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: The sequence number of the ACK segment sent by the client is 2818463619 (No.297 segment). The Acknowledgment field is 1247095791 (No. 297 segment, and 1 greater than the sequence number of the SYNACK segment from the server). This segment doesn't contain any data. In the Flags field of TCP packets, the ACK flag is set to 1 which means it is an ACK packet.

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: Both the client and the server have done the active close. They may decide to close the connection at the same time and they both generate the FIN segment before receiving the FIN segment from the other side. We can determine this by observing No.304 and No.305 segment, which are from client and server respectively. Therefore, the type is Simultaneous close. In 3 Segment and 4 Segment, the FIN segment is from one side, but in Simultaneous close, both client and server are sending independently the FIN segment. And finally, they will give a response ACK(ACK field = FIN packet's Sequence number + 1) to the other side separately, like the No.306 and No.308 segment.

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: Since the transferred data bytes are associated with the sequence number, we can find that the data bytes is 2818463652 – 2818463619 = 33. In fact, the Initial Sequence Number is used to establish connection and it doesn't contain any data, so we should use the next segment (No.297 segment) from client to be the initial sequence number. And when server receives the packets with data from client, it will send ACK segment (ACK field = Seq number from the client packet + data bytes from the client packet). Hence, the final ACK received from server can be used to determine the transferred data bytes. However, in this connection, the final ACK from server is used to determine the FIN from client, so the ACK field should be minus 1 to be used to get the data bytes. Therefore, (the final ACK from the server - 1) - (Initial Sequence Number +1) = data bytes. It is 2818463652 – 2818463619 = 33 bytes.