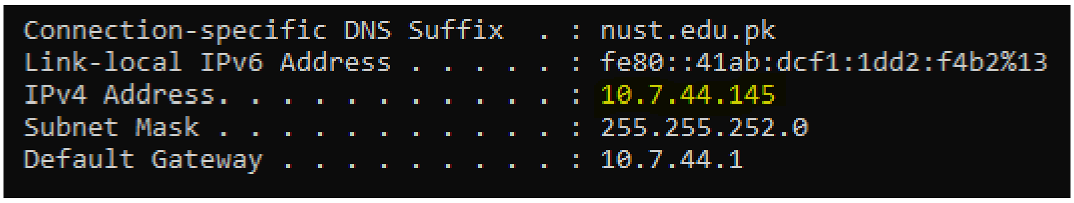
Muhammad Rizwan Khalid
BSCS – 6A
191408



```
Connection-specific DNS Suffix  . : nust.edu.pk
Link-local IPv6 Address . . . . . : fe80::41ab:dcf1:1dd2:f4b2%13
IPv4 Address. . . . . . . . . . . : 10.7.44.145
Subnet Mask . . . . . . . . . . . : 255.255.252.0
Default Gateway . . . . . . . . . : 10.7.44.1
```

Lab Title: *Analysis of TCP packets in Wireshark*

Objective of this lab:

In this lab, we'll investigate the behavior of TCP in detail. We'll do so by analyzing TCP segments sent and received from your computer to a remote server. We'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer.

Instructions:

- *Read carefully before starting the lab.*
- *These exercises are to be done individually.*
- *You are supposed to provide the answers to the questions listed at the end of this document and upload the completed report to your course's LMS site.*
- *Avoid plagiarism by copying from the Internet or from your peers. You may refer to source/ text but you must paraphrase the original work.*

Background:

1. ***The Transmission Control Protocol (TCP) Introduction:***

TCP provides connections between clients and servers. A TCP client establishes a connection with a given server, exchanges data with that server across the connection, and then terminates the connection.

TCP also provides **reliability**. When TCP sends data to the other end, it requires an acknowledgment in return. If an acknowledgment is not received, TCP automatically retransmits the data and waits a longer amount of time. After some number of retransmissions, TCP will give up, with the total amount of time spent trying to send data typically between 4 and 10 minutes (depending on the implementation).

TCP contains algorithms to estimate the **round-trip time** (RTT) between a client and server dynamically so that it knows how long to wait for an acknowledgment. For example, the RTT on a LAN can be milliseconds while across a WAN it can be in seconds. Furthermore, TCP continuously estimates the RTT of a given connection, because the RTT is affected by variations in the network traffic.

TCP also sequences the data by associating a **sequence number** with every byte that it sends. For example, assume an application writes 2,048 bytes to a TCP socket, causing TCP to send two segments, the first containing the data with sequence numbers 1–1,024 and the second containing the data with sequence numbers 1,025–2,048. (A segment is the unit of data that TCP passes to IP.) If the segments arrive out of order, the receiving TCP will reorder the two segments based on their sequence numbers before passing the data to the receiving application. If TCP receives duplicate data from its peer (say the peer thought a segment was lost and retransmitted it, when it wasn't really lost, the network was just overloaded), it can detect that the data has been duplicated (from the sequence numbers), and discard the duplicate data.

In contrast to TCP, there is no reliability provided by UDP. UDP itself does not provide anything like acknowledgments, sequence numbers, RTT estimation, timeouts, or retransmissions. If a UDP datagram is duplicated in the network, two copies can be delivered to the receiving host. Also, if a UDP client sends two datagrams to the same destination, they can be reordered by the network and arrive out of order.

TCP provides **flow control**. TCP always tells its peer exactly how many bytes of data it is willing to accept from the peer at any one time. This is called the advertised window. At any time, the window is the amount of room currently available in the receive buffer, guaranteeing that the sender cannot overflow the receive buffer. The window changes dynamically over time: As data is received from the sender, the window size decreases, but as the receiving application reads data from the buffer, the window size increases. It is possible for the window to reach 0: when TCP's receive buffer for a socket is full and it must wait for the application to read data from the buffer before it can take any more data from the peer.

Finally, a TCP connection is **full-duplex**. This means that an application can send and receive data in both directions on a given connection at any time. This means that TCP must keep track of state information such as sequence numbers and window sizes for each direction of data flow: sending and receiving. After a full-duplex connection is established, it can be turned into a simplex connection if desired.

2. *TCP Connection Establishment and Termination*

2.1 Three-Way Handshake:

Following scenario occurs when a TCP connection is established:

1. The server must be prepared to accept an incoming connection. This is normally done by calling socket, bind, and listen and is called a passive open.
2. The client issues an active open by calling connect. This causes the client TCP to send a "synchronize" (SYN) segment, which tells the server the client's initial sequence number for the data that the client will send on the connection. Normally, there is no data sent with the SYN; it just contains an IP header, a TCP header, and possible TCP options (which we will talk about shortly).
3. The server must acknowledge (ACK) the client's SYN and the server must also send its own SYN containing the initial sequence number for the data that the server will send on the connection. The server sends its SYN and the ACK of the client's SYN in a single segment.
4. The client must acknowledge the server's SYN.

The minimum number of packets required for this exchange is three; hence, this is called TCP's three-way handshake. We show the three segments in Figure 1

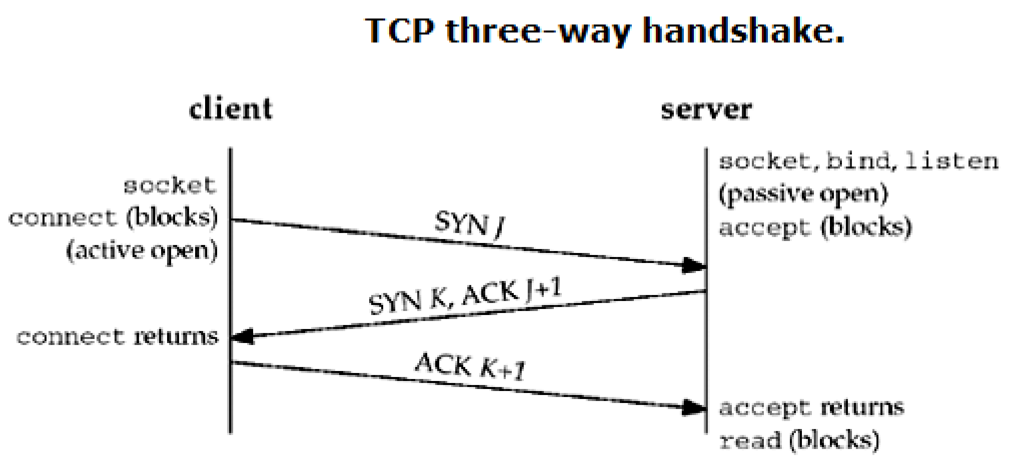


Figure 1

TCP three-way handshake.

We show the client's initial sequence number as J and the server's initial sequence number as K. The acknowledgment number in an ACK is the next expected sequence number for the end sending the ACK. Since a SYN occupies one byte of the sequence number space, the acknowledgment number in the ACK of each SYN is the initial sequence number plus one. Similarly, the ACK of each FIN is the sequence number of the FIN plus one.

An everyday analogy for establishing a TCP connection is the telephone system. The **socket** function is the equivalent of having a telephone to use. **bind** is telling other people your telephone number so that they can call you. **listen** is turning on the ringer so that you will hear when an incoming call arrives. **connect** requires that we know the other person's phone number and dial it. **accept** is when the person being called answers the phone. Having the client's identity returned by accept (where the identify is the client's IP address and port number) is similar to having the caller ID feature show the caller's phone number. One difference, however, is that accept returns the client's identity only after the connection has been established, whereas the caller ID feature shows the caller's phone number before we choose whether to answer the phone or not.

2.2. TCP Connection Termination

While it takes three segments to establish a connection, it takes four to terminate a connection.

1. One application calls close first, and we say that this end performs the active close. This end's TCP sends a FIN segment, which means it is finished sending data.
2. The other end that receives the FIN performs the passive close. The received FIN is acknowledged by TCP. The receipt of the FIN is also passed to the application as an end-of-file (after any data that may have already been queued for the application to receive), since the receipt of the FIN means the application will not receive any additional data on the connection.
3. Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a FIN.
4. The TCP on the system that receives this final FIN (the end that did the active close) acknowledges the FIN.

Since a FIN and an ACK are required in each direction, four segments are normally required. We use the qualifier "normally" because in some scenarios, the FIN in Step 1 is sent with data. Also, the segments in Steps 2 and 3 are both from the end performing the passive close and could be combined into one segment. We show these packets in Figure 2.

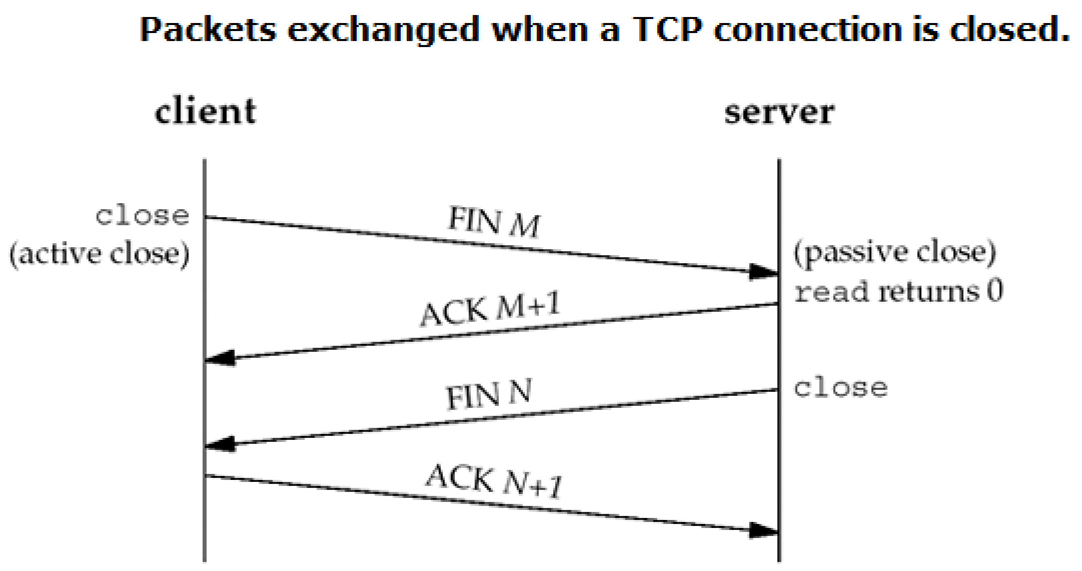


Figure 2 Packets exchanged when a TCP connection is closed.

A FIN occupies one byte of sequence number space just like a SYN. Therefore, the ACK of each FIN is the sequence number of the FIN plus one.

Between Steps 2 and 3 it is possible for data to flow from the end doing the passive close to the end doing the active close. This is called a half open connection.

The sending of each FIN occurs when a socket is closed. We indicated that the application calls close for this to happen, but realize that when a Unix process terminates, either voluntarily (calling exit or having the main function return) or involuntarily (receiving a signal that terminates the process), all open descriptors are closed, which will also cause a FIN to be sent on any TCP connection that is still open.

Although we show the client in Figure 2 performing the active close, either end—the client or the server—can perform the active close. Often the client performs the active close, but with some protocols (notably HTTP), the server performs the active close.

Steps for obtaining credit for this lab.

Do the following:

1. ***Download** files TCPCient.py and TCPServer.py from your LMS site.*

2. ***Edit** the client file to change two things. The serverIP address and the message; as your name.*

3. ***Start up the Wireshark software.***

4. ***Begin packet capture,** select the Capture pull down menu and select Options.*

5. ***Selecting the network interface on which packets would be captured:** You can use most of the default values in this window. The network interfaces (i.e., the physical connections) that your computer has to the network will be shown in the Interface pull down menu at the top of the Capture Options window. Click Start. Packet capture will now begin*

6. ***Run your TCPServer and your edited TCPClient.***

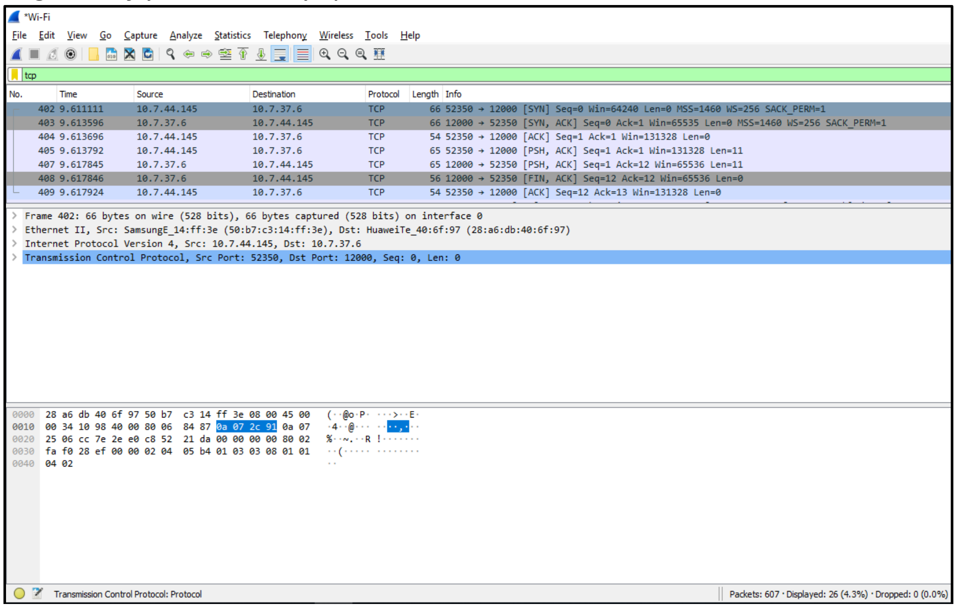
7. ***Stopping the capture and inspecting captured packets:** After you have received a message, stop Wireshark packet capture*

8. ***Filtering:** Filter the TCP packets.*

7. ***Details of a packet:** Select the TCP messages shown in the packet-listing window and*

analyze by looking into the detail of packets pane and answer the questions given at the end of this document.

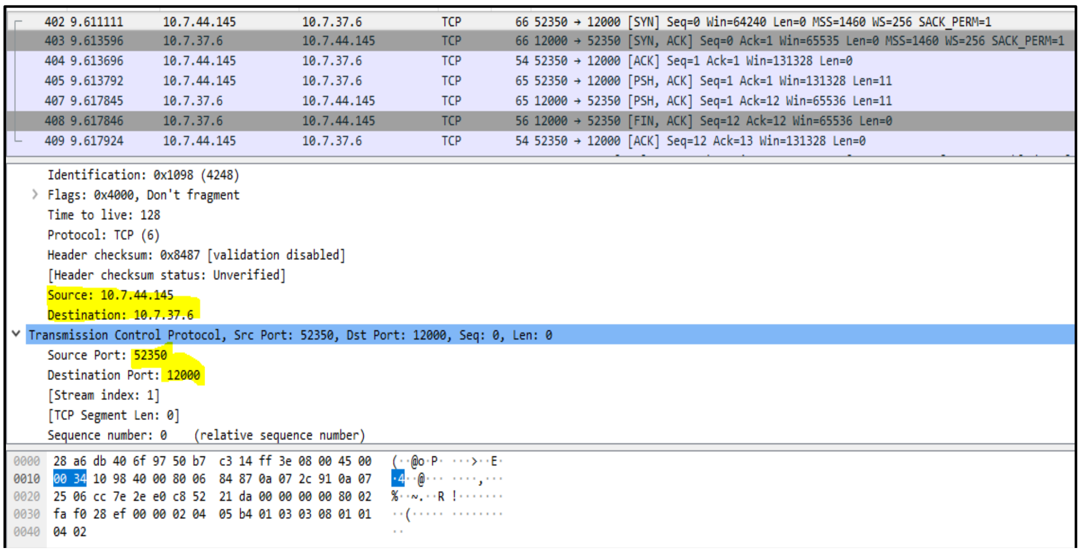
8. ***Obtaining credit for this lab:*** *Now, please proceed to the questions section to answer the questions. You must note down your answers, along with screen shots in this file itself. Please note that you must upload this file (after duly filling in the answers) through the appropriate link at your LMS to obtain credit. Please clarify with your instructor/ lab engineer if you have any queries.*



```
*Wi-Fi
File  Edit  View  Go  Capture  Analyze  Statistics  Telephony  Wireless  Tools  Help
tcp

No.    Time        Source         Destination    Protocol  Length  Info
   402 9.611111    10.7.44.145    10.7.37.6      TCP       66 52350 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
   403 9.613596    10.7.37.6      10.7.44.145    TCP       66 12000 → 52350 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
   404 9.613696    10.7.44.145    10.7.37.6      TCP       54 52350 → 12000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
   405 9.613792    10.7.44.145    10.7.37.6      TCP       65 52350 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=11
   407 9.617845    10.7.37.6      10.7.44.145    TCP       65 12000 → 52350 [PSH, ACK] Seq=1 Ack=12 Win=65536 Len=11
   408 9.617846    10.7.37.6      10.7.44.145    TCP       56 12000 → 52350 [FIN, ACK] Seq=12 Ack=12 Win=65536 Len=0
   409 9.617924    10.7.44.145    10.7.37.6      TCP       54 52350 → 12000 [ACK] Seq=12 Ack=13 Win=131328 Len=0

> Frame 402: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: SamsungE_14:ff:3e (50:b7:c3:14:ff:3e), Dst: HuaweiTe_40:6f:97 (28:a6:db:40:6f:97)
> Internet Protocol Version 4, Src: 10.7.44.145, Dst: 10.7.37.6
> Transmission Control Protocol, Src Port: 52350, Dst Port: 12000, Seq: 0, Len: 0

0000  28 a6 db 40 6f 97 50 b7  c3 14 ff 3e 08 00 45 00   (··@o·P· ···>··E·
0010  00 34 10 98 40 00 80 06  84 87 0a 07 2c 91 0a 07   ·4··@··· ····,···
0020  25 06 cc 7e 2e e0 c8 52  21 da 00 00 00 00 80 02   %··~.··R !·······
0030  fa f0 28 ef 00 00 02 04  05 b4 01 03 03 08 01 01   ··(····· ········
0040  04 02                                              ··

Transmission Control Protocol: Protocol     Packets: 607 · Displayed: 26 (4.3%) · Dropped: 0 (0.0%)
```

Questions:

1. **What are the IP addresses and TCP port numbers used by the client and the server?**

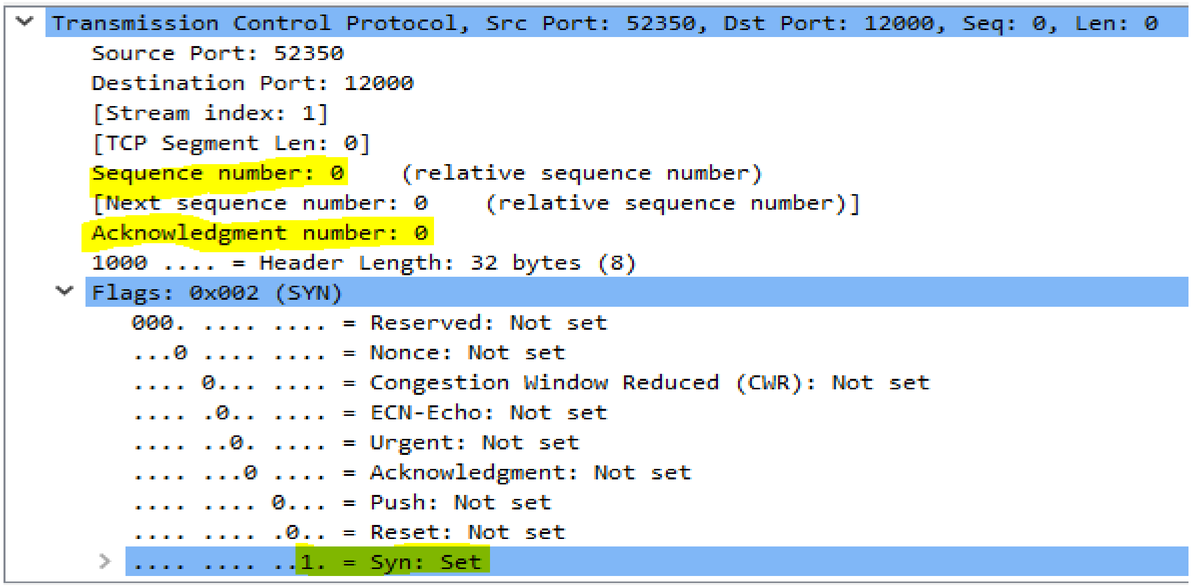


```
   402 9.611111    10.7.44.145    10.7.37.6      TCP    66 52350 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
   403 9.613596    10.7.37.6      10.7.44.145    TCP    66 12000 → 52350 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
   404 9.613696    10.7.44.145    10.7.37.6      TCP    54 52350 → 12000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
   405 9.613792    10.7.44.145    10.7.37.6      TCP    65 52350 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=11
   407 9.617845    10.7.37.6      10.7.44.145    TCP    65 12000 → 52350 [PSH, ACK] Seq=1 Ack=12 Win=65536 Len=11
   408 9.617846    10.7.37.6      10.7.44.145    TCP    56 12000 → 52350 [FIN, ACK] Seq=12 Ack=12 Win=65536 Len=0
   409 9.617924    10.7.44.145    10.7.37.6      TCP    54 52350 → 12000 [ACK] Seq=12 Ack=13 Win=131328 Len=0

    Identification: 0x1098 (4248)
  > Flags: 0x4000, Don't fragment
    Time to live: 128
    Protocol: TCP (6)
    Header checksum: 0x8487 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.7.44.145
    Destination: 10.7.37.6
v Transmission Control Protocol, Src Port: 52350, Dst Port: 12000, Seq: 0, Len: 0
    Source Port: 52350
    Destination Port: 12000
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)

0000  28 a6 db 40 6f 97 50 b7  c3 14 ff 3e 08 00 45 00   (··@o·P· ···>··E·
0010  00 34 10 98 40 00 80 06  84 87 0a 07 2c 91 0a 07   ·4··@··· ····,···
0020  25 06 cc 7e 2e e0 c8 52  21 da 00 00 00 00 80 02   %··~.··R !·······
0030  fa f0 28 ef 00 00 02 04  05 b4 01 03 03 08 01 01   ··(····· ········
0040  04 02                                              ··
```

Client IP: 10.7.44.145, Server IP: 10.7.37.6

Server Port: 12000, Client Port: 52350

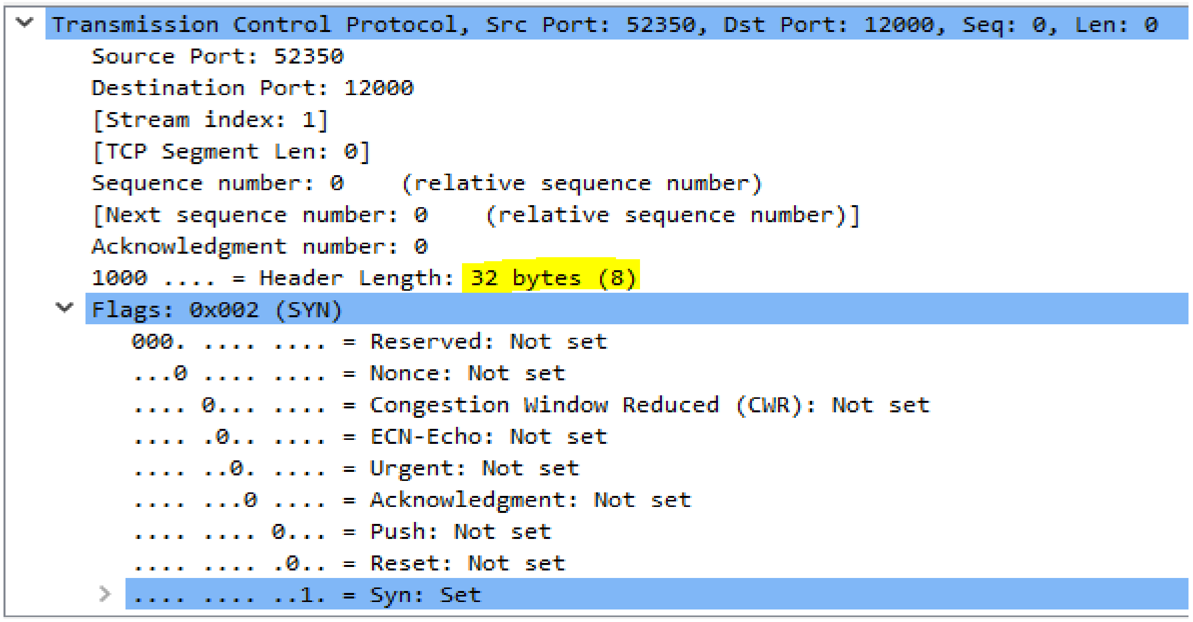
2. **What are the SEQ and ACK Nos of the TCP SYN segment that is used to initiate the TCP connection between the client and server?**



```
Transmission Control Protocol, Src Port: 52350, Dst Port: 12000, Seq: 0, Len: 0
    Source Port: 52350
    Destination Port: 12000
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
```

For initializing the TCP connection, First TCP SYN segment is used by client. As from the figure, it can be seen that SEQ has value of 0. ACK number has the value of 0. While the SYN is set 1.

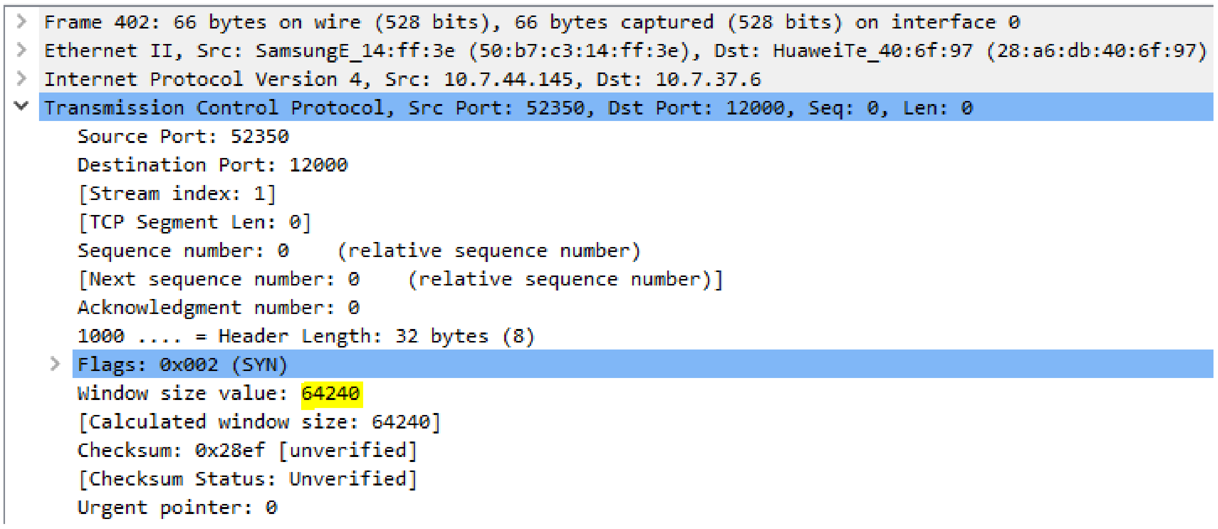
3. **What is the header length of TCP used for this connection?**



```
Transmission Control Protocol, Src Port: 52350, Dst Port: 12000, Seq: 0, Len: 0
    Source Port: 52350
    Destination Port: 12000
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
```

Header Length for this connection is 32 bytes as seen in the above figure. This header contains various information like flags, source port, destination port, window size and checksum etc.

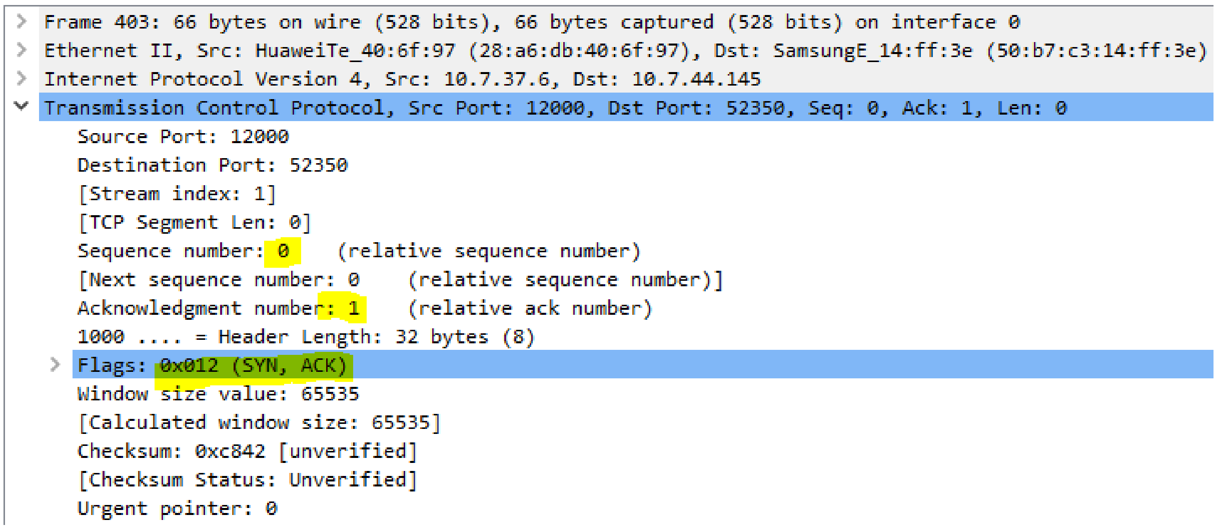
4. **What is the minimum amount of available buffer space advertised by the client and the server for this connection?**



```
> Frame 402: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: SamsungE_14:ff:3e (50:b7:c3:14:ff:3e), Dst: HuaweiTe_40:6f:97 (28:a6:db:40:6f:97)
> Internet Protocol Version 4, Src: 10.7.44.145, Dst: 10.7.37.6
v Transmission Control Protocol, Src Port: 52350, Dst Port: 12000, Seq: 0, Len: 0
     Source Port: 52350
     Destination Port: 12000
     [Stream index: 1]
     [TCP Segment Len: 0]
     Sequence number: 0    (relative sequence number)
     [Next sequence number: 0    (relative sequence number)]
     Acknowledgment number: 0
     1000 .... = Header Length: 32 bytes (8)
   > Flags: 0x002 (SYN)
     Window size value: 64240
     [Calculated window size: 64240]
     Checksum: 0x28ef [unverified]
     [Checksum Status: Unverified]
     Urgent pointer: 0
```

For this connection the minimum amount of available buffer is governed by the window size and that is 64240 bytes as seen in the above picture.

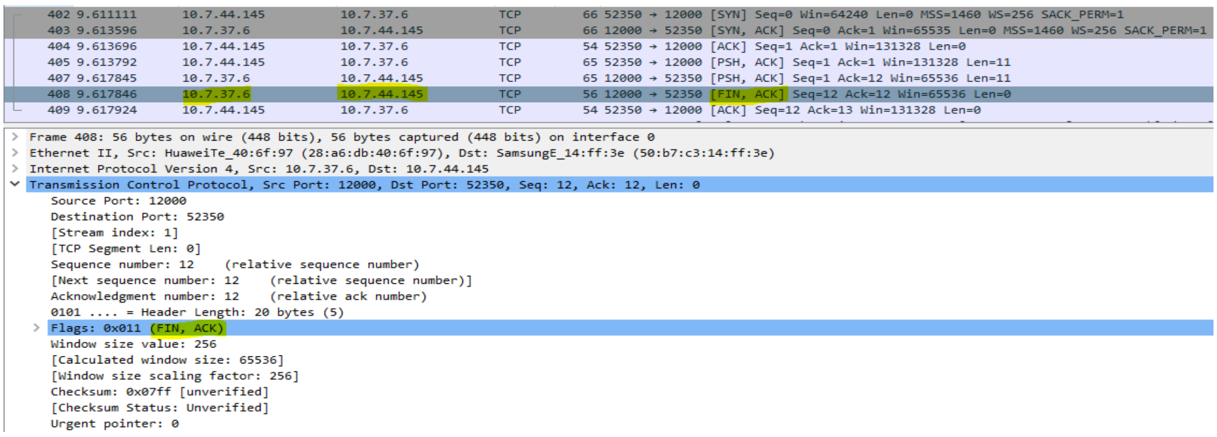
5. **What is the sequence number of the SYNACK segment sent by server to the client computer in reply to the SYN? What is the value of the ACK field in the SYNACK segment? How did the server determine that value?**



```
> Frame 403: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: HuaweiTe_40:6f:97 (28:a6:db:40:6f:97), Dst: SamsungE_14:ff:3e (50:b7:c3:14:ff:3e)
> Internet Protocol Version 4, Src: 10.7.37.6, Dst: 10.7.44.145
v Transmission Control Protocol, Src Port: 12000, Dst Port: 52350, Seq: 0, Ack: 1, Len: 0
     Source Port: 12000
     Destination Port: 52350
     [Stream index: 1]
     [TCP Segment Len: 0]
     Sequence number: 0    (relative sequence number)
     [Next sequence number: 0    (relative sequence number)]
     Acknowledgment number: 1    (relative ack number)
     1000 .... = Header Length: 32 bytes (8)
   > Flags: 0x012 (SYN, ACK)
     Window size value: 65535
     [Calculated window size: 65535]
     Checksum: 0xc842 [unverified]
     [Checksum Status: Unverified]
     Urgent pointer: 0
```

In response to the SYN segment, SYNACK segment is sent by the server to the client. It contains the SEQ value 0, ACK value 1 and SYN value 1. Server calculates the ACK value by adding one to SEQ value, thus giving one in this case. Through ACK server is saying that it is ready to get connected.

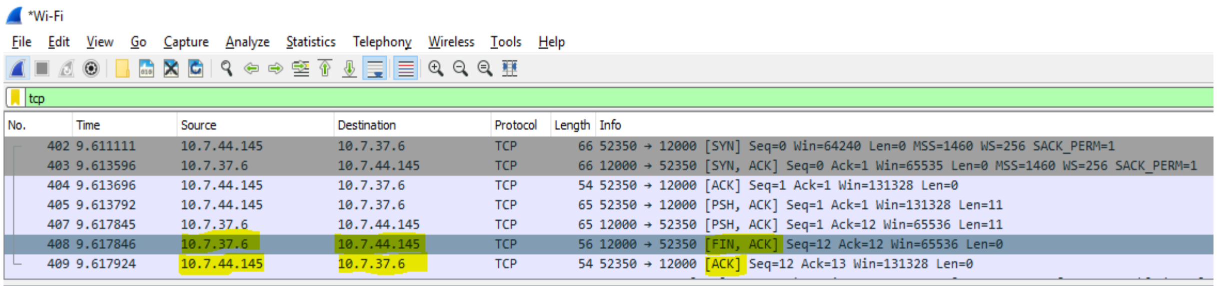
6. **Who has done the active close? Client or the server? How you have determined this?**



```
402 9.611111   10.7.44.145   10.7.37.6     TCP   66 52350 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
403 9.613596   10.7.37.6     10.7.44.145   TCP   66 12000 → 52350 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
404 9.613696   10.7.44.145   10.7.37.6     TCP   54 52350 → 12000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
405 9.613792   10.7.44.145   10.7.37.6     TCP   65 52350 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=11
407 9.617845   10.7.37.6     10.7.44.145   TCP   65 12000 → 52350 [PSH, ACK] Seq=1 Ack=12 Win=65536 Len=11
408 9.617846   10.7.37.6     10.7.44.145   TCP   56 12000 → 52350 [FIN, ACK] Seq=12 Ack=12 Win=65536 Len=0
409 9.617924   10.7.44.145   10.7.37.6     TCP   54 52350 → 12000 [ACK] Seq=12 Ack=13 Win=131328 Len=0

> Frame 408: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
> Ethernet II, Src: HuaweiTe_40:6f:97 (28:a6:db:40:6f:97), Dst: SamsungE_14:ff:3e (50:b7:c3:14:ff:3e)
> Internet Protocol Version 4, Src: 10.7.37.6, Dst: 10.7.44.145
v Transmission Control Protocol, Src Port: 12000, Dst Port: 52350, Seq: 12, Ack: 12, Len: 0
    Source Port: 12000
    Destination Port: 52350
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 12    (relative sequence number)
    [Next sequence number: 12    (relative sequence number)]
    Acknowledgment number: 12    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x011 (FIN, ACK)
    Window size value: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0x07ff [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
```

As seen in the above picture that source ip: 10.7.37.6 has sent the FIN/ACK segment and in my case 10.7.37.6 is the server ip. So, server has done the active close.

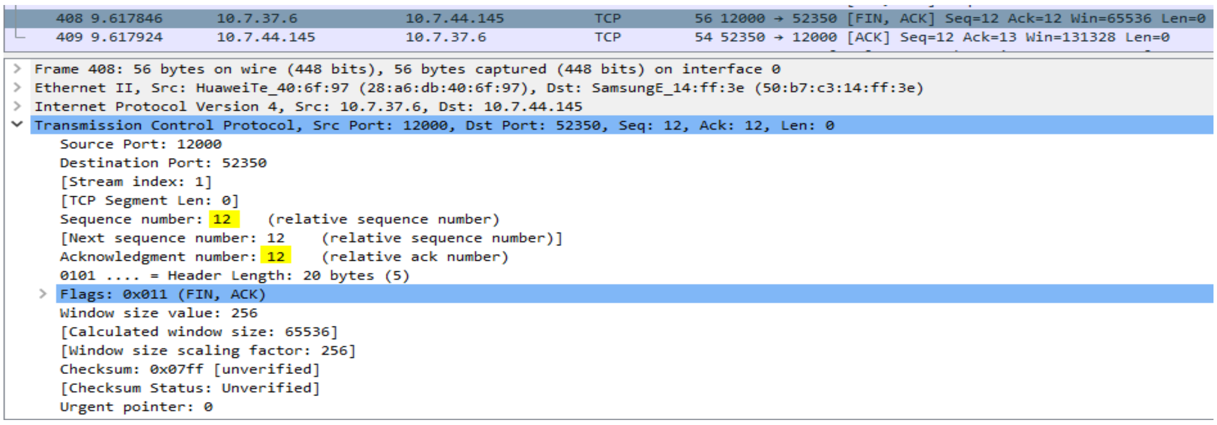
7. **What type of closure has been performed? 3 segment (FIN/FIN-ACK/ACK) or four segments (FIN/ACK/FIN/ACK) or simultaneous close?**


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp

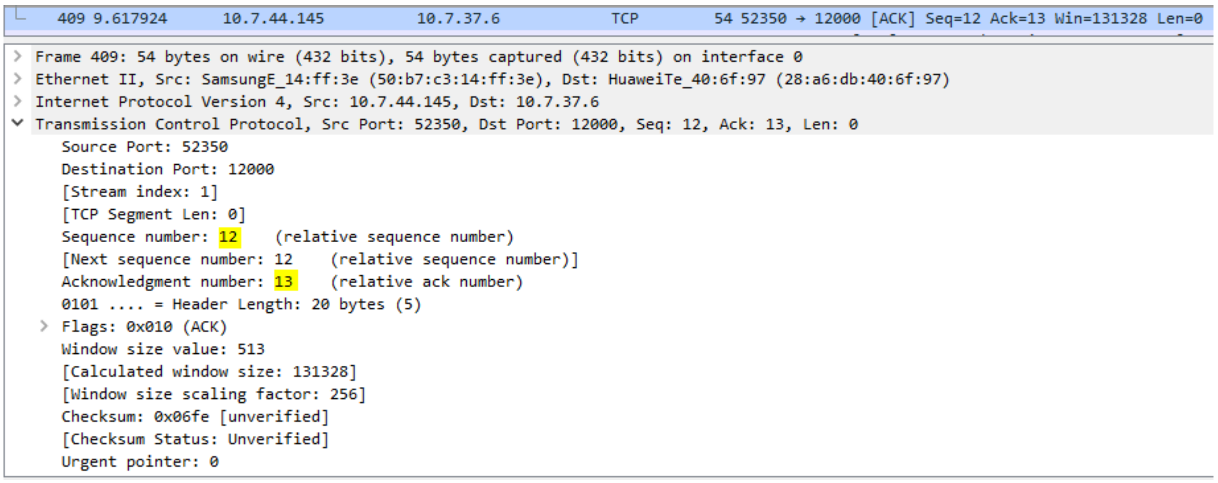
```
No.  Time       Source        Destination   Protocol Length Info
402 9.611111   10.7.44.145   10.7.37.6     TCP   66 52350 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
403 9.613596   10.7.37.6     10.7.44.145   TCP   66 12000 → 52350 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
404 9.613696   10.7.44.145   10.7.37.6     TCP   54 52350 → 12000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
405 9.613792   10.7.44.145   10.7.37.6     TCP   65 52350 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=11
407 9.617845   10.7.37.6     10.7.44.145   TCP   65 12000 → 52350 [PSH, ACK] Seq=1 Ack=12 Win=65536 Len=11
408 9.617846   10.7.37.6     10.7.44.145   TCP   56 12000 → 52350 [FIN, ACK] Seq=12 Ack=12 Win=65536 Len=0
409 9.617924   10.7.44.145   10.7.37.6     TCP   54 52350 → 12000 [ACK] Seq=12 Ack=13 Win=131328 Len=0
```

3 Segment closure has been performed. As seen from the above picture that the server has done the active close and later on client has acknowledged it.

8. **What are SEQ and ACK Nos for all the segments used for the connection closure?**



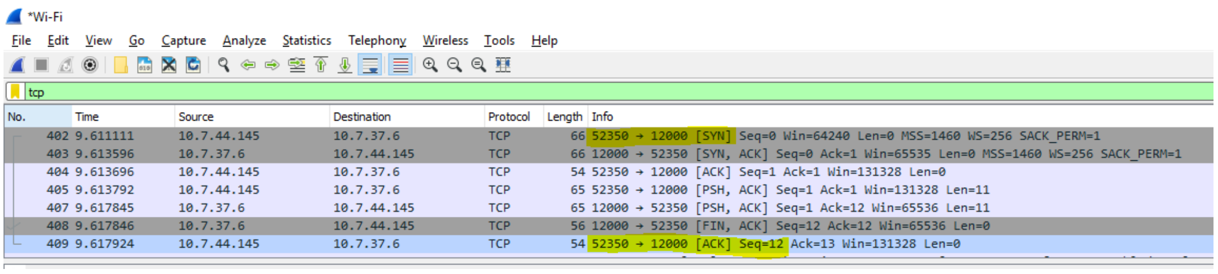
```
408 9.617846   10.7.37.6     10.7.44.145   TCP   56 12000 → 52350 [FIN, ACK] Seq=12 Ack=12 Win=65536 Len=0
409 9.617924   10.7.44.145   10.7.37.6     TCP   54 52350 → 12000 [ACK] Seq=12 Ack=13 Win=131328 Len=0

> Frame 408: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
> Ethernet II, Src: HuaweiTe_40:6f:97 (28:a6:db:40:6f:97), Dst: SamsungE_14:ff:3e (50:b7:c3:14:ff:3e)
> Internet Protocol Version 4, Src: 10.7.37.6, Dst: 10.7.44.145
v Transmission Control Protocol, Src Port: 12000, Dst Port: 52350, Seq: 12, Ack: 12, Len: 0
    Source Port: 12000
    Destination Port: 52350
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 12    (relative sequence number)
    [Next sequence number: 12    (relative sequence number)]
    Acknowledgment number: 12    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x011 (FIN, ACK)
    Window size value: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0x07ff [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
```



```
409 9.617924    10.7.44.145    10.7.37.6    TCP    54 52350 → 12000 [ACK] Seq=12 Ack=13 Win=131328 Len=0

> Frame 409: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
> Ethernet II, Src: SamsungE_14:ff:3e (50:b7:c3:14:ff:3e), Dst: HuaweiTe_40:6f:97 (28:a6:db:40:6f:97)
> Internet Protocol Version 4, Src: 10.7.44.145, Dst: 10.7.37.6
v Transmission Control Protocol, Src Port: 52350, Dst Port: 12000, Seq: 12, Ack: 13, Len: 0
    Source Port: 52350
    Destination Port: 12000
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 12    (relative sequence number)
    [Next sequence number: 12    (relative sequence number)]
    Acknowledgment number: 13    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window size value: 513
    [Calculated window size: 131328]
    [Window size scaling factor: 256]
    Checksum: 0x06fe [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
```

As shown in the first figure above that first segment (FIN/ACK) for closing the TCP connection has a SEQ value and ACK value of 12. This segment was sent from server to the client. While the second figure shows the SEQ number 12 and ACK number 13 for the ACK segment and this segment was sent by client to the server.

9. **How many (data) bytes in total have been transferred from the client to the server and from the server to the client during the whole connection? What relationship this has with the initial SEQ and the final ACK received from the other side?**


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp

```
No.    Time        Source         Destination    Protocol  Length  Info
  402 9.611111    10.7.44.145    10.7.37.6      TCP         66 52350 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
  403 9.613596    10.7.37.6      10.7.44.145    TCP         66 12000 → 52350 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
  404 9.613696    10.7.44.145    10.7.37.6      TCP         54 52350 → 12000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
  405 9.613792    10.7.44.145    10.7.37.6      TCP         65 52350 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=11
  407 9.617845    10.7.37.6      10.7.44.145    TCP         65 12000 → 52350 [PSH, ACK] Seq=1 Ack=12 Win=65536 Len=11
  408 9.617846    10.7.37.6      10.7.44.145    TCP         56 12000 → 52350 [FIN, ACK] Seq=12 Ack=12 Win=65536 Len=0
  409 9.617924    10.7.44.145    10.7.37.6      TCP         54 52350 → 12000 [ACK] Seq=12 Ack=13 Win=131328 Len=0
```

Data bytes + SEQ numbers + ACKs are the total data/bytes have been transferred bidirectionaly between client and server. It adds up to 24. 22 bytes for the data and 2 bytes for the connection purposes. The first SEQ value initializes the connection between the client and server and final ACK closed the connection between them.