***Name: Mujtaba Shahid Faizi***

***Regn No: 131818***

***Current IP Address: 10.3.31.233***

***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.

1. ***TCP Connection Establishment and Termination***

#### *1.1 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.

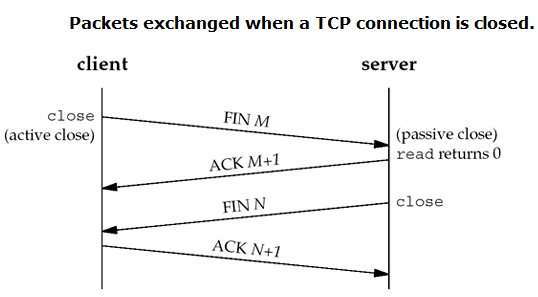
An everyday analogy for establishing a TCP connection is the telephone system. The **socket** function is the equivalent of having a telephone to use. **bind** is telling other people your telephone number so that they can call you. **listen** is turning on the ringer so that you will hear when an incoming 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.

#### *1.2. 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.

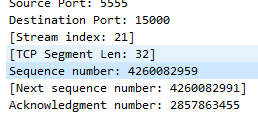
Although we show the client in Figure 2 performing the active close, either end—the client or the server—can perform the active close. Often the client performs the active close, but with some protocols (notably HTTP), the server performs the active close.

*Steps for obtaining credit for this lab.*

**Questions:**

1. **What are the SEQ and ACK Nos of the TCP SYN, SYN/ACK and ACK segments that are used to initiate the TCP connection between the client and server?**





1. **What is the minimum amount of available buffer space advertised by the client and the server for this connection? Does this value change during the data transfer?**

Buffer=Window Size-> 64240



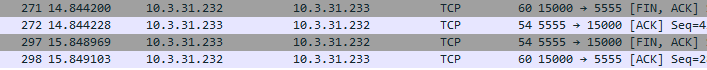
1. **Who has done the active close? Client or the server? How you have determined this?**

The active close would be done by the server side since the fin has been generated by the server.



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

It’s a four segmented closure.



1. **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?**





Data bytes->32 sent

Sequence number->4260082959

Acknowledgment number-> seqnum + data\_bytes

->4260082991