**TCP Design**

* **TCP Manager:** Keeps a hash set of bound sockets so as to avoid binding to a port number that is already in use. A hash table of listening sockets (port number -> TCP Socket) is also kept separately so as to facilitate constant time look up of a requested listening socket in SYN requests received. Finally, another hash table of connections is also maintained for constant time access to connections established. The connections table is mapped from a string (formatted as [destIP:destPort, srcIP:srcPort]) to the connection socket. The connection table enhances routing of packet to the correct connection after examining its header specifying the destination and source (IP:port) pairs.
* **TCP Socket Buffer:** The designed buffer is circular and can act as either receiving buffer or sending buffer. The buffer keeps track of base and length indices. The base index keeps track of the first unsent byte (to network layer in case of a sending buffer or to application layer in case of a receiving buffer). The length index similarly keeps track of the last byte received from the network layer or application layer. The buffer is made circular so that after running off the last index in the byte array, it goes back to the first index.
* **TCP Socket:** Can be a listening socket on server side or a connection socket on either the sender (client side) or the receiver (server side). The socket keeps only one TCP buffer because each socket is designed to be single stream so that it can’t be receiving and sending data simultaneously. The socket keeps a request queue of connections (sockets) and backlog for each server side connection. On server side, a listening socket creates a new connection socket on receipt of a SYN request. The new connection socket is added to the request queue awaiting the accept() call. TCP socket also manages hand shaking, connection termination and retransmission of lost packets. Moreover, it is responsible for handling both flow control and congestion control.
* **Flow control:** This is implemented by making use of the advertised window field of the segments. A receiver advertises the amount of free space it has left in its buffer (receiver’s window size) by using the window field in the ACK segments it sends. The TCP buffer on the sender side determines the number of packets to be sent by taking the minimum of the receiver’s window size and the sender’s congestion window size.
* **Congestion control:** My implementation followed the FSM described by Kurose & Ross in figure 3.52 (page 275) of Computer Networking. There are three states: slow start, congestion avoidance and fast recovery. TCP socket is initialized with slow start state with congestion window set to one packet. The congestion window size keeps increasing linearly by one unit with every receipt of new ACK. When size of congestion window increases beyond the threshold value, the FSM goes to congestion avoidance mode whereby the congestion window is increased much slowly. Three consecutive duplicate ACKs trigger the fast recovery mode. Timeouts take the congestion window size back to one packet. The timeouts used for retransmission of packets is also changed dynamically as per the description in section 3.5.3 (page 238-241). Round trip times of sampled packets (sampleRTT) are used to dynamically update the value of timeout interval. The following are the equations used each time to update the timeout interval:

EstimatedRTT = (1 – α) \* EstimatedRTT + α \* sampleRTT

devRTT = (1 - β) \* devRTT + β \* | sampleRTT – estimatedRTT |

timeoutInterval = estimatedRTT + 4 \* devRTT

where α = 0.125 and β = 0.25 as per section 3.5.3 of Computer Networks.

**Discussion Questions**

1. A random value chosen as the initial sequence number is better. This is because it will enable different packets from concurrent connections to have unique identifiers. Thus packet identification is less prone to error since different connections will each have a different set of sequence numbers for its packets.
2. Under a SYN flood attack, my current implementation will open up multiple connections with the requester for each SYN request until the limit imposed by backlog is reached after which SYN requests will be dropped. To overcome this kind of attack, we can filter out the SYN requests before creating connections. More specifically, the TCP manager that maintains a table of connections can first check if a connection with the SYN requester’s IP address already exists. If so, the manager can be designed to limit the maximum number of connections that a given IP address is allowed. However, if the attacker uses a distributed SYN flood from different IP addresses, then this system is not likely to work.
3. In my current implementation, if the sender never closes the connection, the receiver’s connection will not be closed either. This is because the receiver terminates the connection only on receipt of FIN from sender. To handle FIN attacks, we can use timeouts after which the receiver will automatically close the connection if no data is being received from the sender. Alternatively, each connection can be designed as a session with a fixed duration after which the sender is forced to handshake again. A failed handshake would lead to the termination of connection.
4. The size of receive buffer should depend on the speed of the application reading from the buffer and the bandwidth of the network (more specifically, the bandwidth delay product). Picking the size of receive buffer this way will ensure that it is efficiently able to keep up with the incoming data from the network vis a vis draining from the application reading from it. The buffer size is typically set to about 64kB but should be adjusted accordingly.