# TCP Tahoe

I have implemented TCP Tahoe as my reliable data transfer protocol.
I am sending a file from client to server as follows:

- The file is first divided into byte array of size 128 bytes.
- I have defined a TCP header class which includes fields like checksum, fin flag which indicates end of file, sequence number ,payload and acknowledgement number.
- The byte array is assigned to the payload field and stored in a Tree Map along with its sequence number.
- Checksum is then computed on each payload and assigned to the checksum field.
- The object is serialized and then sent over the network to the server in the form of packets.
- The fin flag is set to 1 when

The Receiver or the server deserializes the object ,retrieves the payload from it and writes it to a file.

The above process takes place using UDP protocol. But in order to make data transfer reliable I am implementing the above process by using TCP Tahoe mechanism.
I have created 4 classes in all.
UDPClient : Client which sends file to the server. It has two threads. One thread receives acknowledgements and other thread sends packets.

UDPServer: Two threads. One accepts incoming packets and the other sends acknowledgements.

TCP: Serializable class which includes TCP header field.
ACK : Serializable class which includes acknowledgement to be sent.
**TCP Tahoe:**

- I am sending the packets over the network along with its sequence number ,checksum and acknowledgement number.
- My implementation of TCP Tahoe makes use of two variables sstresh and cwnd. sstresh is the maximum limit upto which there is no congestion in the network.
- Slow Start: cwnd is 1 and only a single packet is sent .The client waits till it receives an acknowledgement from the server. As soon as it receives an ack it increments cwnd by 1 and now it can send 2 packets at a time. Basically, packets equal to the window or cwnd size can be sent together. As and when the client receives an acknowledgement it keeps increasing its congestion window by 1.

- I am running a for loop which starts from the last acknowledged packet to the sliding window size.(i.e. every time when I am receiving an acknowledgement I am sliding my window to the 1st packet number in the congestion window and also increase the size of congestion window by 1 for correctly received acknowledgements. )
- This continues till my cwnd reaches sstresh. As cwnd goes beyond sstresh we come to know that there is congestion in the network.
- Now instead of increasing cwnd by 1 for each correct acknowledgement cwnd is now increased by cwnd+(1/cwnd). This process is called congestion avoidance.
- As we are sending packets over a network there is a chance that the packets might get lost in the network. To handle packet loss, My receiver /Server keeps track of expected packets and compares each received packet with the expected packet number. If it received what it had expected it will store the packet in a TreeMap and will send an acknowledgement to the client with the next expected packet number.
- At the Server two TreeMaps are maintained. One keeps track of the received packets and the other buffers the payload along with its sequence number so that the file can be regenerated.
- If the Server receives packet number 0 it will buffer the data of the packet in one TreeMap and store the sequence number along with its Boolean value(true as and when packets are received) in another TreeMap named keepTrack. It will then store the next expected packet (In the above case it will be packet number 1) with Boolean value false in the keepTrack treeMap.
- This sequence (expected packet) number is then sent as an acknowledgement to the Client.
- But if it receives a packet which was not the packet it expected it sends an acknowledgement to the client with the packet number it had expected by traversing the keepTrack TreeMap and finding the lowest key with Boolean value false.
- It keeps sending duplicate acknowledgements till it receives the lost packet.
- The Client on the other side retransmits the lost packet only after three duplicate acknowledgements or after timeout.
- After retransmitting the packet sstresh is reduced to cwnd/2 and cwnd is set to 1.

Steps to execute the program:

Command Line options:
-c: to run as client
-s: to run as server.
-quiet: displays only the checksum calculated at both the ends.
-timeout: if not provided 1000ms by default

Example Test Case:

First Compile:

```
 javac fcntcp.java
```

Server Side:

```
 java fcntcp –s 3000
```

Client Side:

```
Java fcntcp –c –f filename –timeout 1000 serveraddress 3000
```