This sequence diagram was generated with EventStudio System Designer (http://www.EventHelix.com/EventStudio).

TCP Slow Start and Congestion Avoidance lower the data throughput drastically when segment loss is detected. Fast Retransmit and Fast Recovery have been designed to speed up the recovery of the connection, without compromising its congestion avoidance characteristics.

Fast Retransmit and Recovery detect a segment loss via duplicate acknowledgements. When a segment is lost, TCP at the receiver will keep sending ack segments indicating the next expected sequence number. This sequence number would correspond to the lost segment. If only one segment is lost, TCP will keep generating acks for the following segments. This will result in the transmitter getting duplicate acks (i.e. acks with the same ack sequence number)

Socket initialization

Server socket initialization

Server Socket

create

Server Application creates a Socket

Closed

The Socket is created in Closed state

seq_num = 100

Server sets the initial sequence number to 100

Passive_Open

Server application has initiated a passive open. In this mode, the socket does not attempt to establish a TCP connection. The socket listens for TCP connection request from clients

Listen

Socket transitions to the Listen state

Server awaits client socket connections.

Client initiated three way handshake to establish a TCP connection

SYN

src = Client_Port,
dst = Server_Port,
seq_num = 0

SYN TCP segment is received by the server

SYN+ACK

src = Server_Port,
dst = Client_Port,
seq_num = 100,
ack_num = 1,
window = 65535

Server sets the SYN and the ACK bits in the TCP header. Server sends its initial sequence number as 100. Server also sets its window to 65535 bytes. i.e. Server has buffer space for 65535 bytes of data. Also note that the ack sequence numer is set to 1. This signifies that the server expects a next byte sequence number of 1

SYN Received

Now the server transitions to the SYN Received state

ACK

src = Client_Port,
dst = Server_Port,
ack_num = 101,
window = 5000

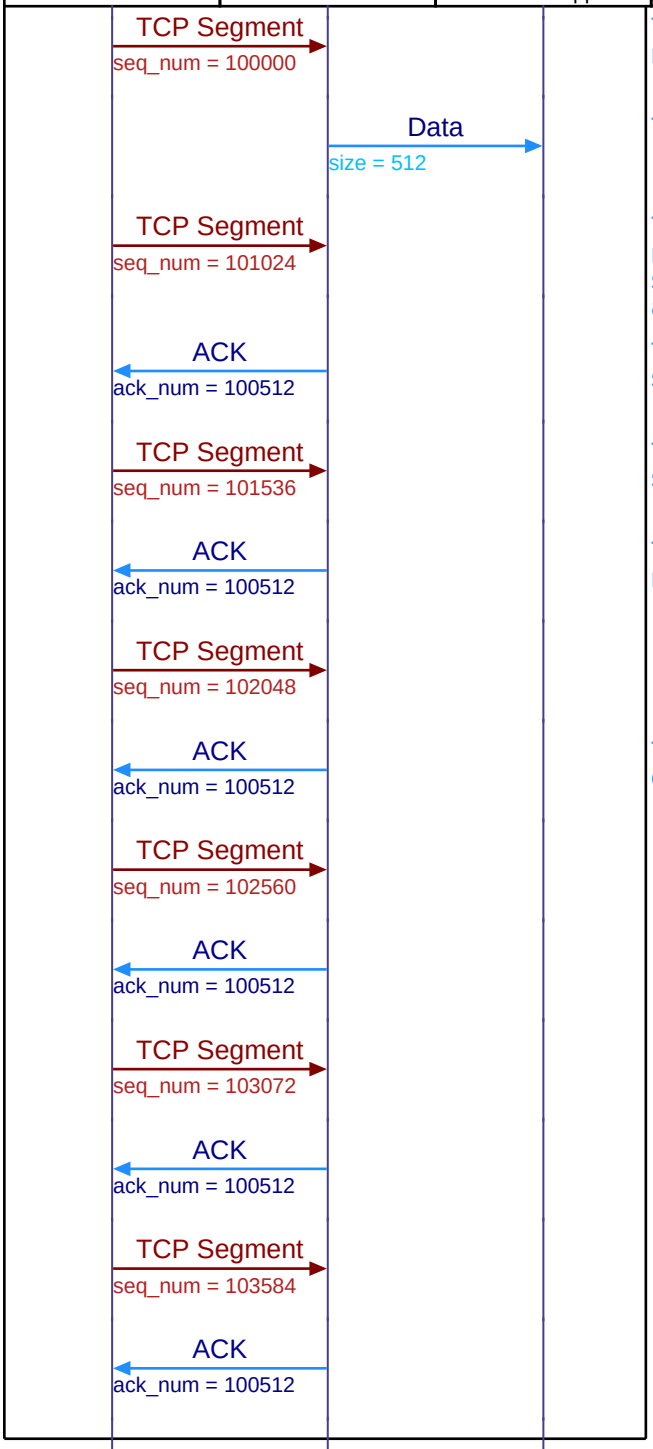
Server receives the TCP ACK segment

Established

Now the server too moves to the Established state

TCP Connection begins with slow start. The congestion window grows from an initial 512 bytes to 70000 bytes

Loss of a TCP segment



TCP segment (start sequence number = 100000) is delivered to the receiver

TCP passes 512 bytes of data to the higher layer

TCP Segment with start sequence number 101024 is received. TCP realizes that a segment has been missed. TCP buffers the out of sequence segment as TCP cannot deliver out of sequence data to the application.

TCP sends an acknowledgement to the Sender with the next expected sequence number set to 100512.

TCP receives the next segment. This and the following out of sequence segments will be buffered by TCP.

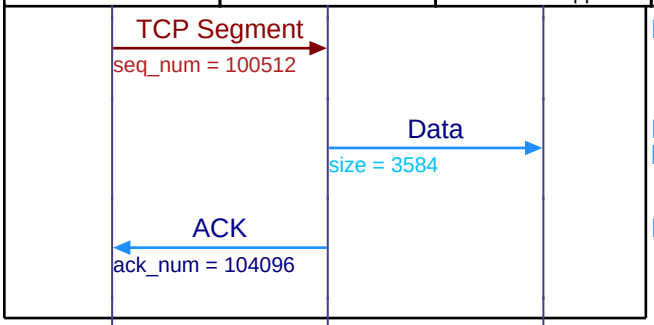
TCP sends another acknowledgement with the next expected sequence number still set to 100512. This is a duplicate acknowledgement

TCP keeps acknowledging the received segments with the next expected sequence number as 100512

Fast Retransmit: TCP receives duplicate acks and it decides to retransmit the segment, without waiting for the segment timer to expire. This speeds up recovery of the lost segment

Fast Recovery

Fast Recovery: Once the lost segment has been transmitted, TCP tries to maintain the current data flow by not going back to slow start. TCP also adjusts the window for all segments that have been buffered by the receiver.

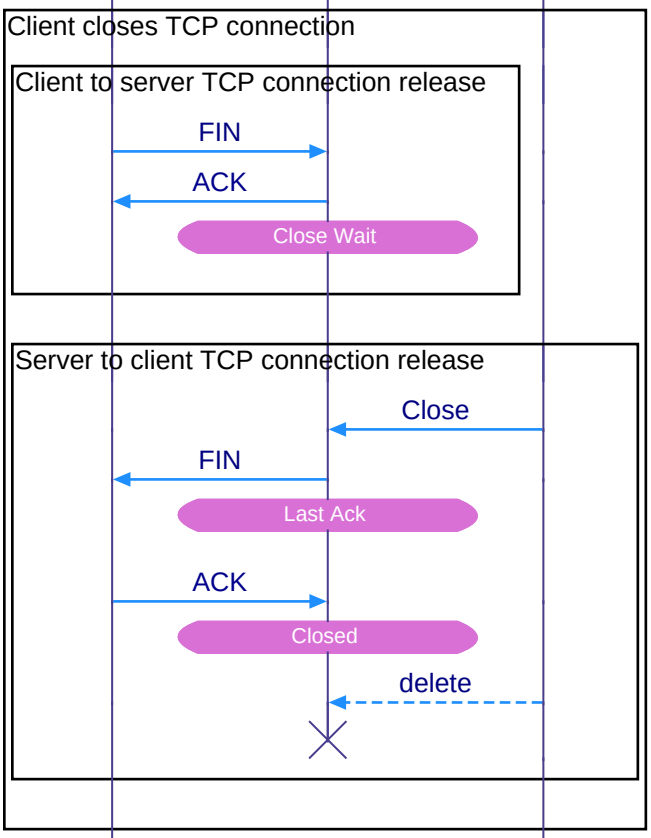


Finally, the retransmitted segment is delivered to the server

Now TCP can pass the just received missing segment and all the buffered segments to the application layer

Now TCP acknowledges all the segments that it had buffered

Congestion Avoidance



Server receives the FIN

Server responds back with ACK to acknowledge the FIN

Server changes state to Close Wait. In this state the server waits for the server application to close the connection

Server application closes the TCP connection

FIN is sent out to the client to close the connection

Server changes state to Last Ack. In this state the last acknowledgement from the client will be received

Server receives the ACK

Server moves the connection to closed state