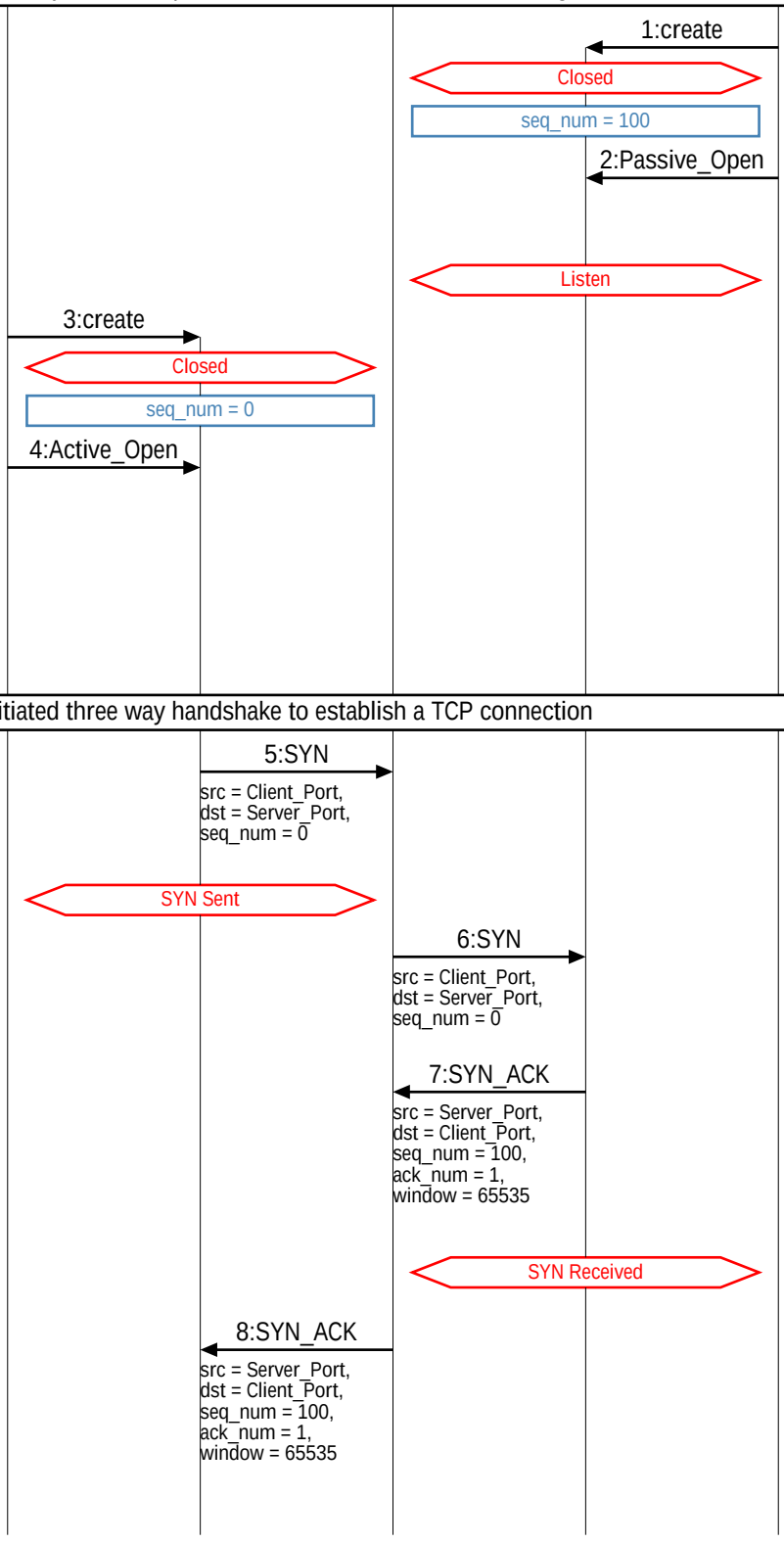
LEG: About TCP

TCP (Transmission Control Protocol) provides a reliable end to end service that delivers packets over the Internet. Packets are delivered in sequence without loss or duplication.

This sequence diagram describes the setup and release of a TCP connection. Datagram handling with a maximum segment size of 512 is also shown. The byte level sequence numbers and TCP Ack handling is also shown.



Server Application creates a Socket

The Socket is created in Closed state

Server sets the initial sequence number to 100

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

Socket transitions to the Listen state

Client Application creates Socket

The socket is created in the Closed state

Initial sequence number is set to 0

Application wishes to communicate with a destination server using a TCP connection. The application opens a socket for the connection in active mode. In this mode, a TCP connection will be attempted with the server.
Typically, the client will use a well known port number to communicate with the remote Server. For example, HTTP uses port 80.

LEG: Client initiates TCP connection

Client initiated three way handshake to establish a TCP connection

Client sets the SYN bit in the TCP header to request a TCP connection. The sequence number field is set to 0. Since the SYN bit is set, this sequence number is used as the initial sequence number

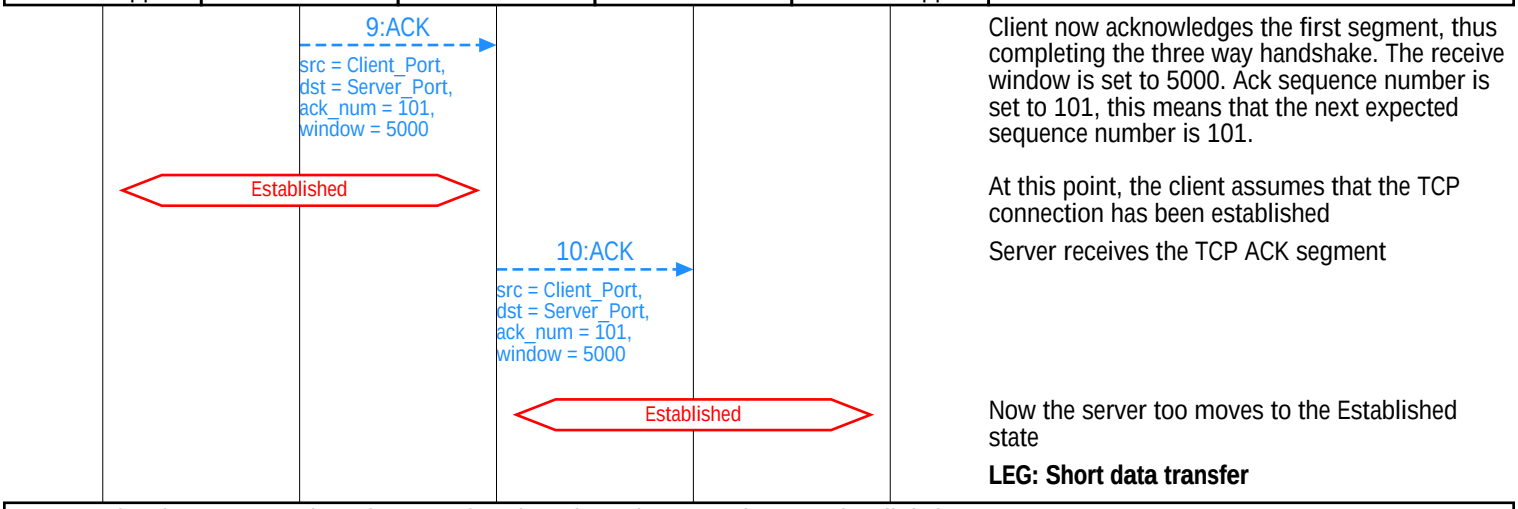
Socket transitions to the SYN Sent state

SYN TCP segment is received by the server

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

Now the server transitions to the SYN Received state

Client receives the SYN_ACK TCP segment



Client now acknowledges the first segment, thus completing the three way handshake. The receive window is set to 5000. Ack sequence number is set to 101, this means that the next expected sequence number is 101.

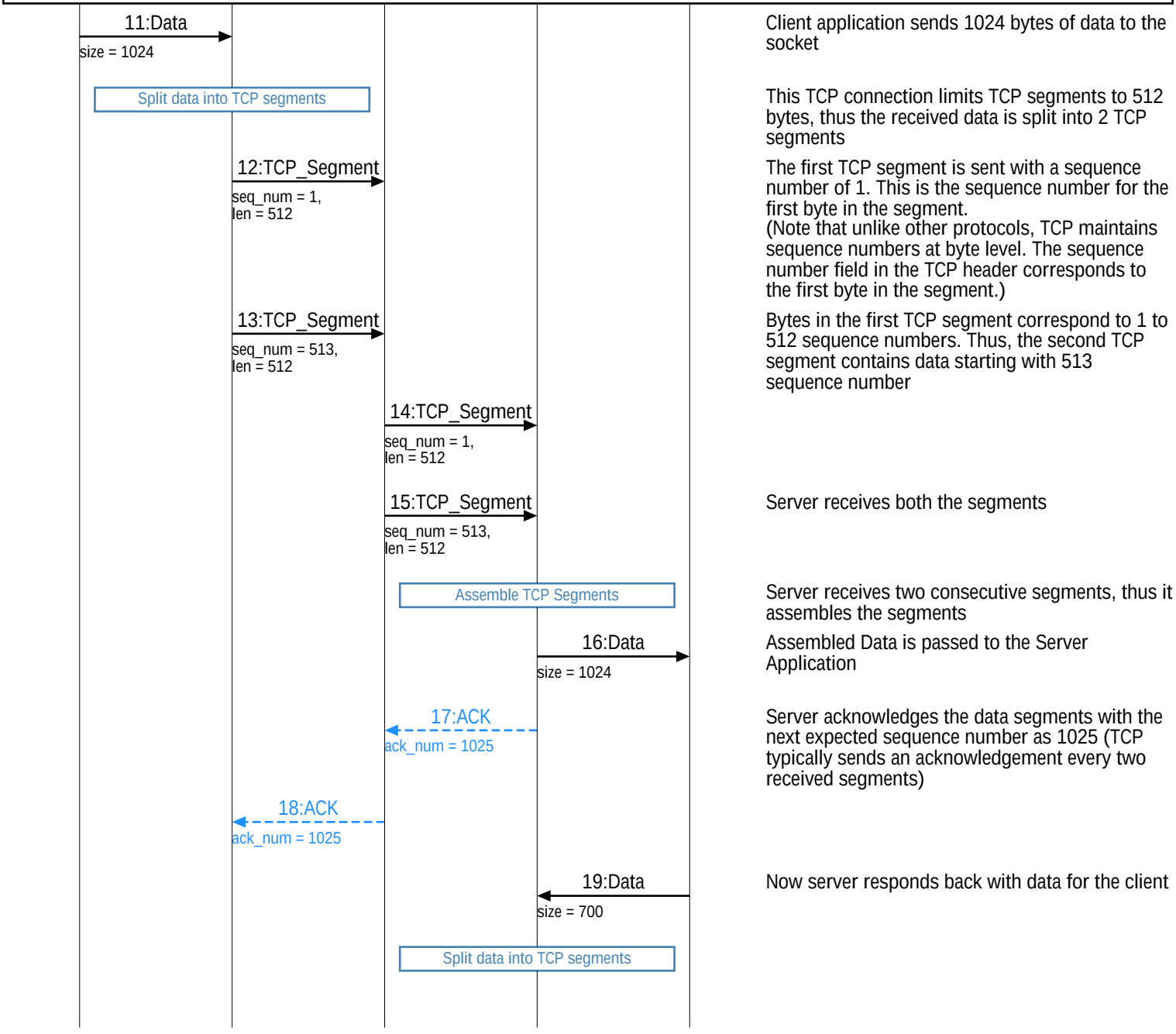
At this point, the client assumes that the TCP connection has been established

Server receives the TCP ACK segment

Now the server too moves to the Established state

LEG: Short data transfer

Data transfer phase: Here a short data transfer takes place, thus TCP slow start has little impact.



Client application sends 1024 bytes of data to the socket

This TCP connection limits TCP segments to 512 bytes, thus the received data is split into 2 TCP segments

The first TCP segment is sent with a sequence number of 1. This is the sequence number for the first byte in the segment.
(Note that unlike other protocols, TCP maintains sequence numbers at byte level. The sequence number field in the TCP header corresponds to the first byte in the segment.)

Bytes in the first TCP segment correspond to 1 to 512 sequence numbers. Thus, the second TCP segment contains data starting with 513 sequence number

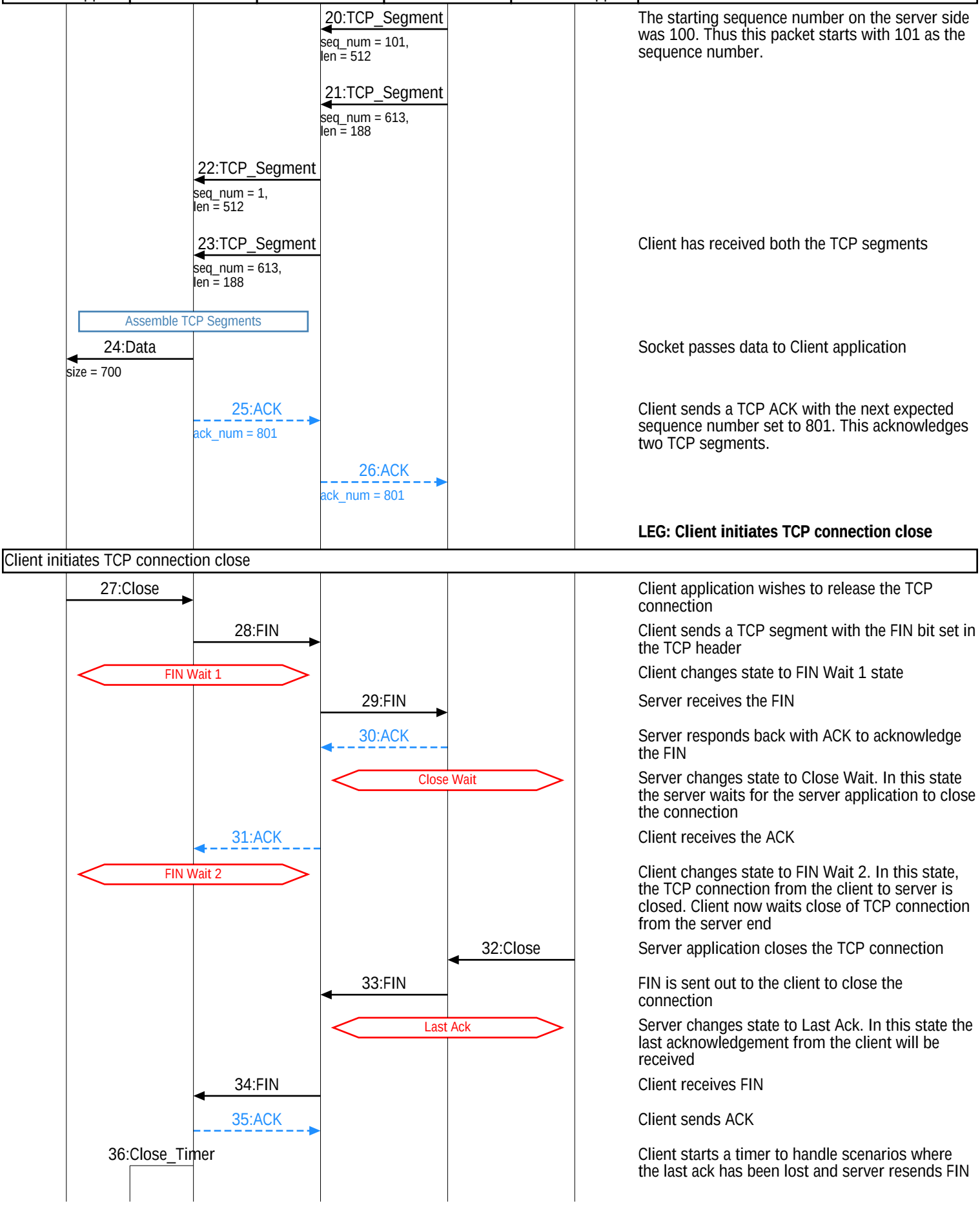
Server receives both the segments

Server receives two consecutive segments, thus it assembles the segments

Assembled Data is passed to the Server Application

Server acknowledges the data segments with the next expected sequence number as 1025 (TCP typically sends an acknowledgement every two received segments)

Now server responds back with data for the client



The starting sequence number on the server side was 100. Thus this packet starts with 101 as the sequence number.

Client has received both the TCP segments

Socket passes data to Client application

Client sends a TCP ACK with the next expected sequence number set to 801. This acknowledges two TCP segments.

LEG: Client initiates TCP connection close

Client application wishes to release the TCP connection

Client sends a TCP segment with the FIN bit set in the TCP header

Client changes state to FIN Wait 1 state

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

Client receives the ACK

Client changes state to FIN Wait 2. In this state, the TCP connection from the client to server is closed. Client now waits close of TCP connection from the server end

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

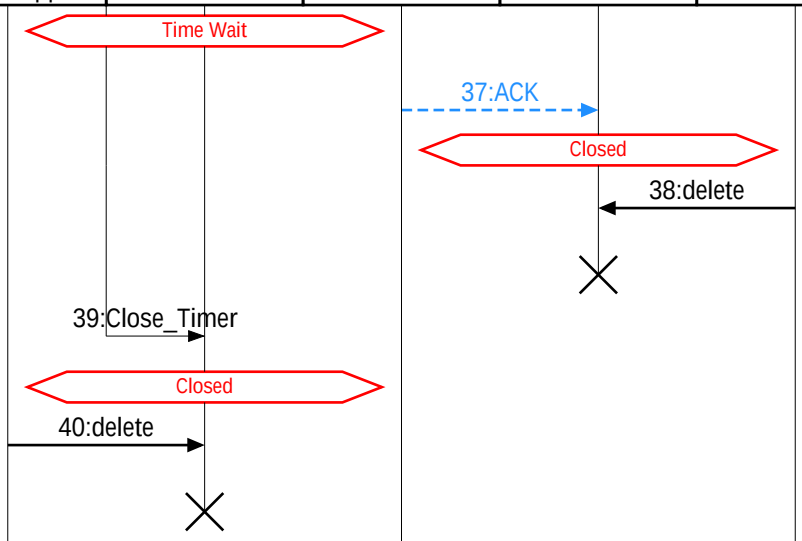
Client receives FIN

Client sends ACK

Client starts a timer to handle scenarios where the last ack has been lost and server resends FIN

TCP - Transmission Control Protocol (TCP Basic Handling)

Client Node		Internet	Server Node		EventStudio System Designer 4.0
Client		Net	Server		14-Jul-07 09:40 (Page 4)
Client App	Client Socket	Network	Server Socket	Server App	



Client waits in Time Wait state to handle a FIN retry

Server receives the ACK

Server moves the connection to closed state

Close timer has expired. Thus the client end connection can be closed too.