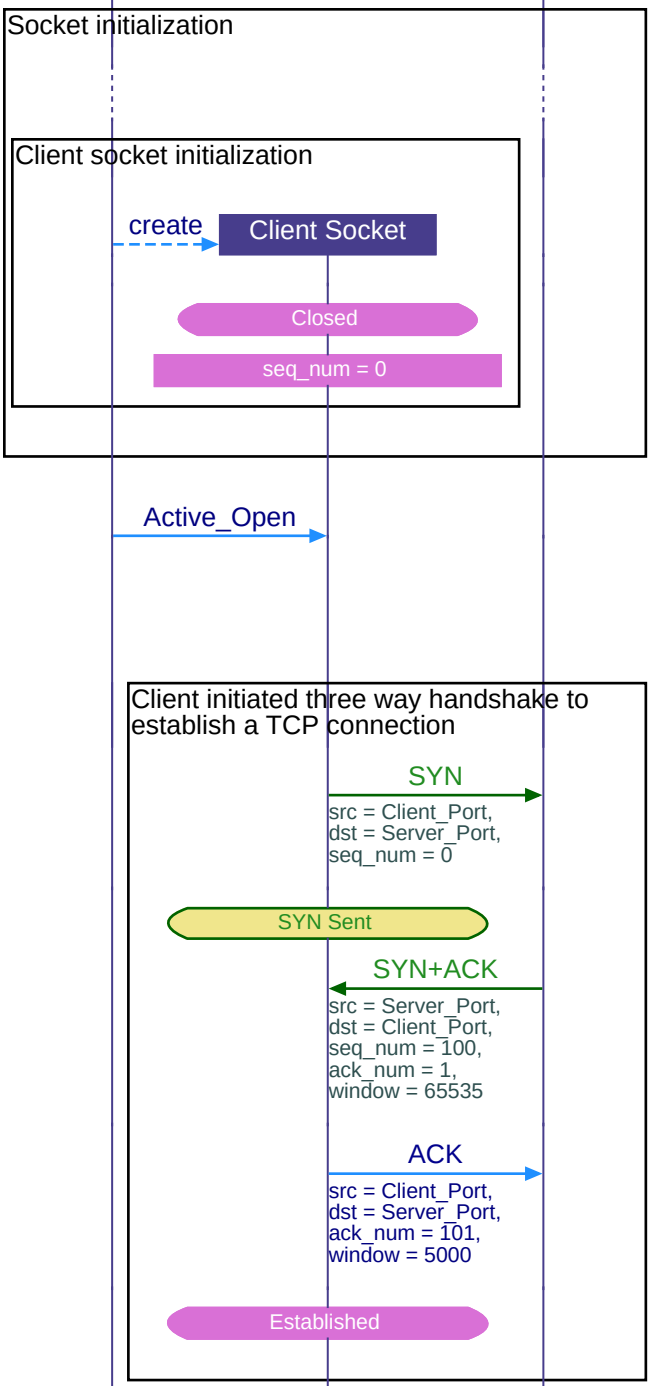
This sequence diagram was generated with EventStudio System Designer (http://www.EventHelix.com/EventStudio).

TCP is an end to end protocol which operates over the heterogeneous Internet. TCP has no advance knowledge of the network characteristics, thus it has to adjust its behavior according to the current state of the network. TCP has built in support for congestion control. Congestion control ensures that TCP does not pump data at a rate higher than what the network can handle.

In this sequence diagram we will analyse "Slow start", an important part of the congestion control mechanisms built right into TCP. As the name suggests, "Slow Start" starts slowly, increasing its window size as it gains confidence about the networks throughput.



Server awaits client socket connections.

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.

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

Client receives the "SYN+ACK" TCP segment

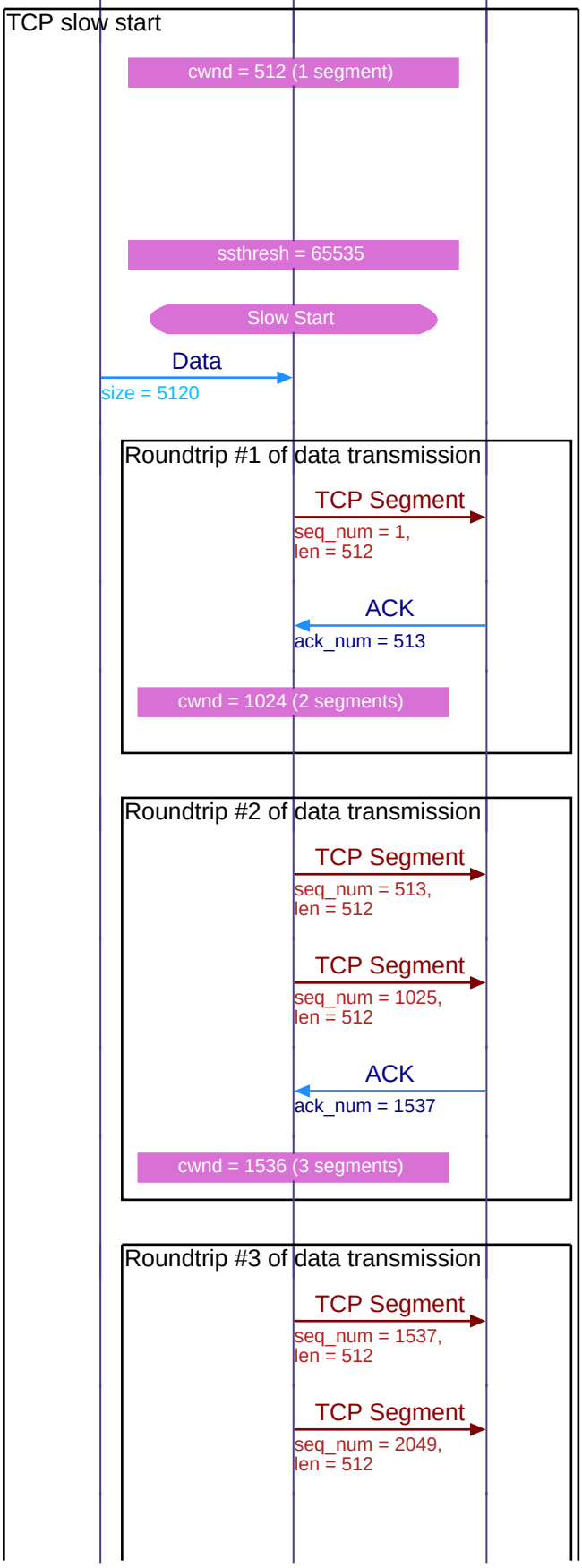
Client now acknowledges the first segment, thus completing the three way handshake. The receive window is set to 5000. Ack sequence number is set to 101, this means that the next expected sequence number is 101.

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

A TCP connection starts in the "Slow Start" state. In this state, TCP adjusts its transmission rate based on the rate at which the acknowledgements are received from the other end.

TCP Slow start is implemented using two variables, viz cwnd (Congestion Window)and ssthresh (Slow Start Threshold).

cwnd is a self imposed transmit window restriction at the sender end. cwnd will increase as TCP gains more confidence on the network's ability to handle traffic. ssthresh is the threshold for determining the point at which TCP exits slow start. If cwnd increases beyond ssthresh, the TCP session in that direction is considered to be out of slow start phase



Client maintains a congestion window (cwnd). Initially the window is set to lower of the maximum TCP segment size and receiver's allowed window size. In most cases the segment size is smaller than receiver window, thus cwnd is set to the maximum TCP segment size (512 in this example)
Note here that cwnd implements a transmitter end flow control. The receiver advertised window implements a receiver enforced flow control.

TCP connections start with ssthresh set to 64K. This variable will be used to determine the point at which TCP exits slow start

Client end TCP connection moves to slow start state

Client application sends 5120 bytes of data to the socket

The first TCP segment is sent with a sequence number of 1. This is the sequence number for the first byte in the segment.

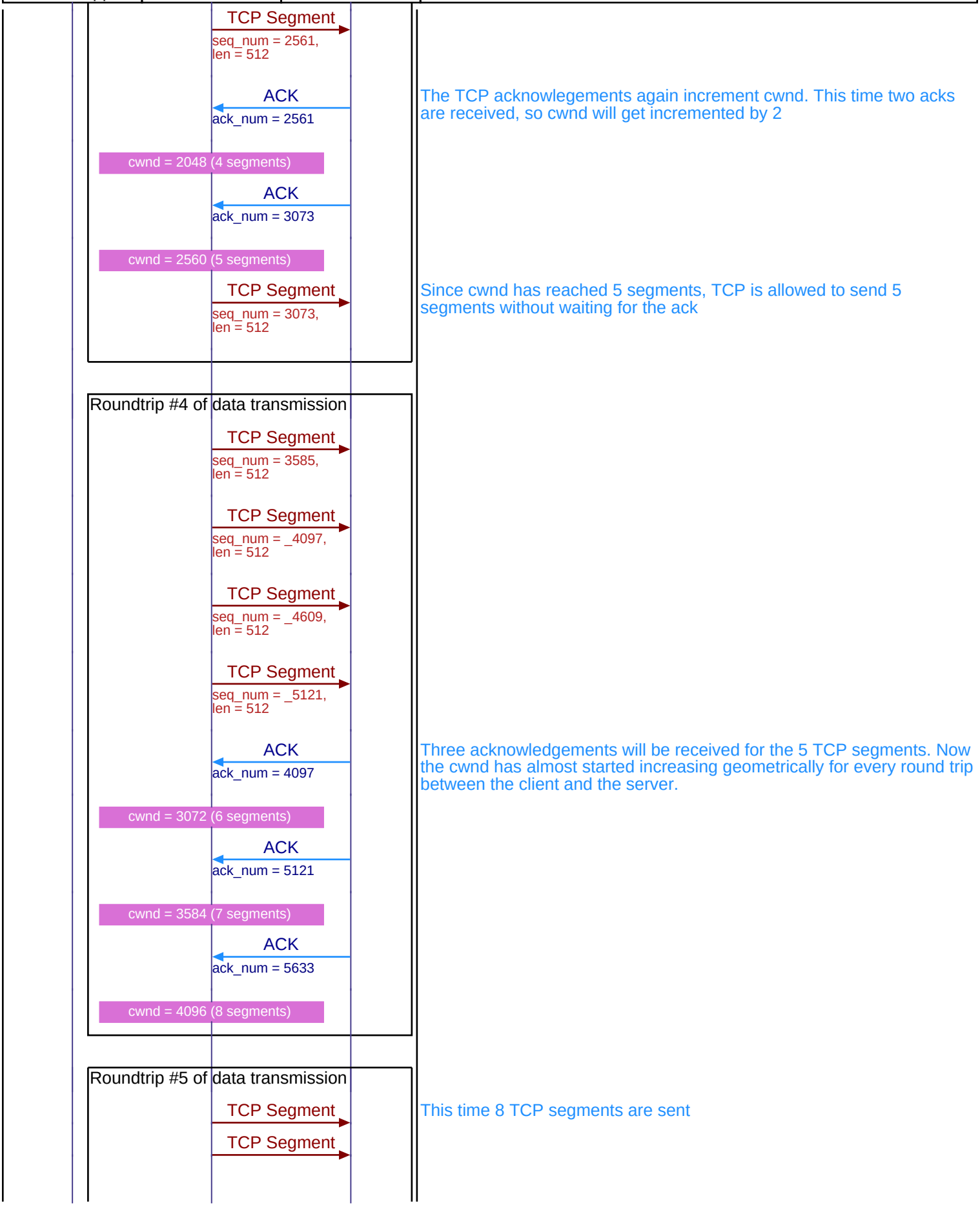
Client receives the acknowledgement for the first TCP data segment

As the TCP session is in slow start, receipt of an acknowledgement increments the congestion window by one 1 segment.

Since the congestion window has increased to 2, TCP can now send two segments without waiting for an ack

Receipt for ack again moves the congestion window

Now three segments can be sent without waiting for an ack

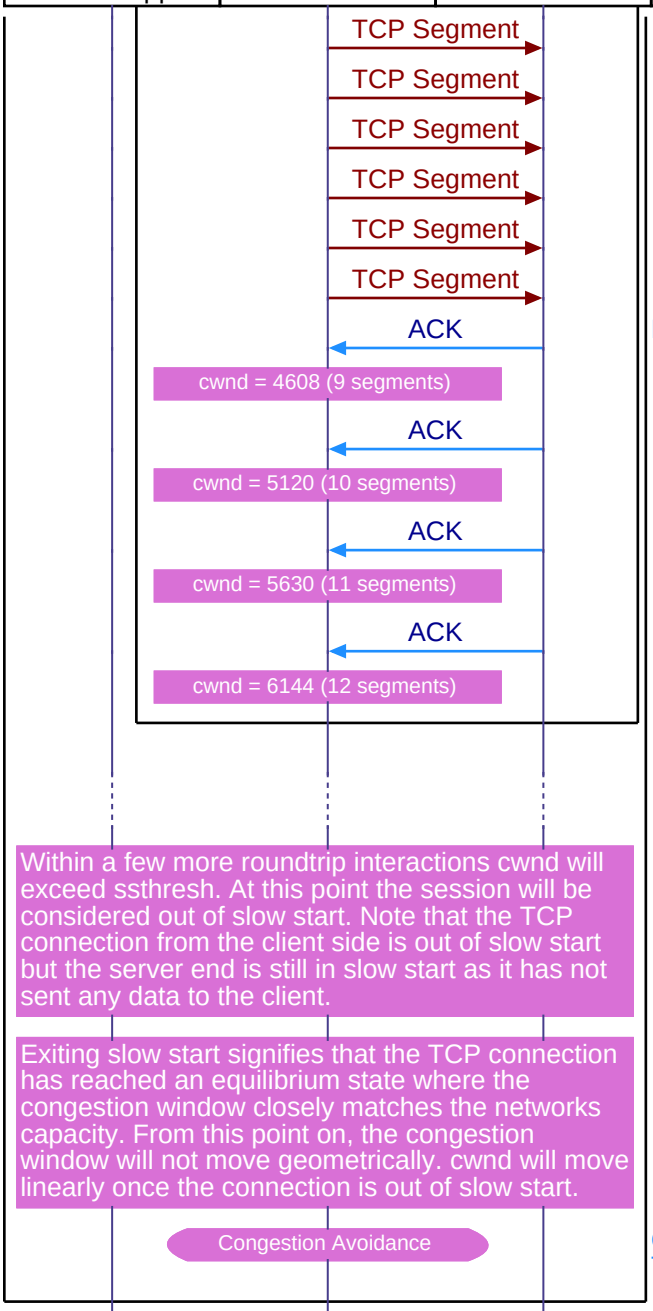


The TCP acknowlegements again increment cwnd. This time two acks are received, so cwnd will get incremented by 2

Since cwnd has reached 5 segments, TCP is allowed to send 5 segments without waiting for the ack

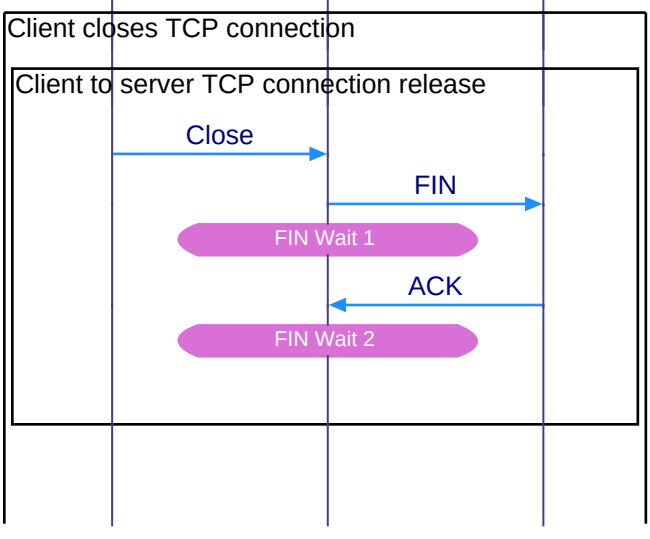
Three acknowledgements will be received for the 5 TCP segments. Now the cwnd has almost started increasing geometrically for every round trip between the client and the server.

This time 8 TCP segments are sent



Now four acks will be received, thus moving cwnd even more quickly

Once slow start ends, the session enters congestion avoidance state. This will be discussed in a subsequent article.



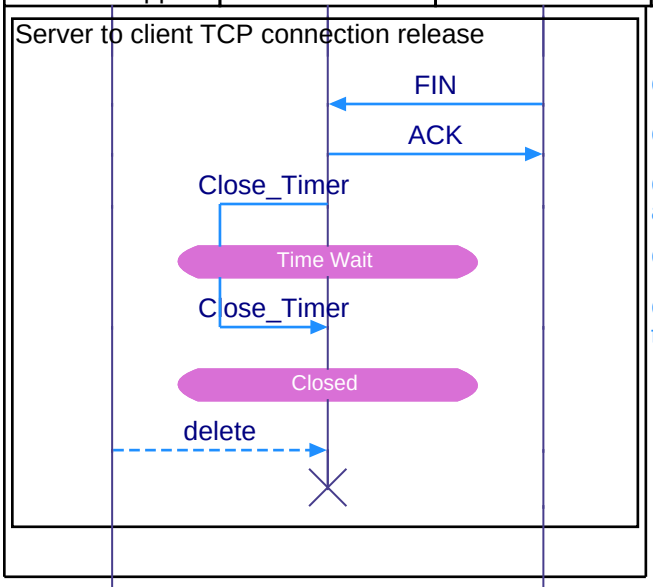
Client application wishes to release the TCP connection

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

Client changes state to FIN Wait 1 state

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



Client receives FIN

Client sends ACK

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

Client waits in Time Wait state to handle a FIN retry

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