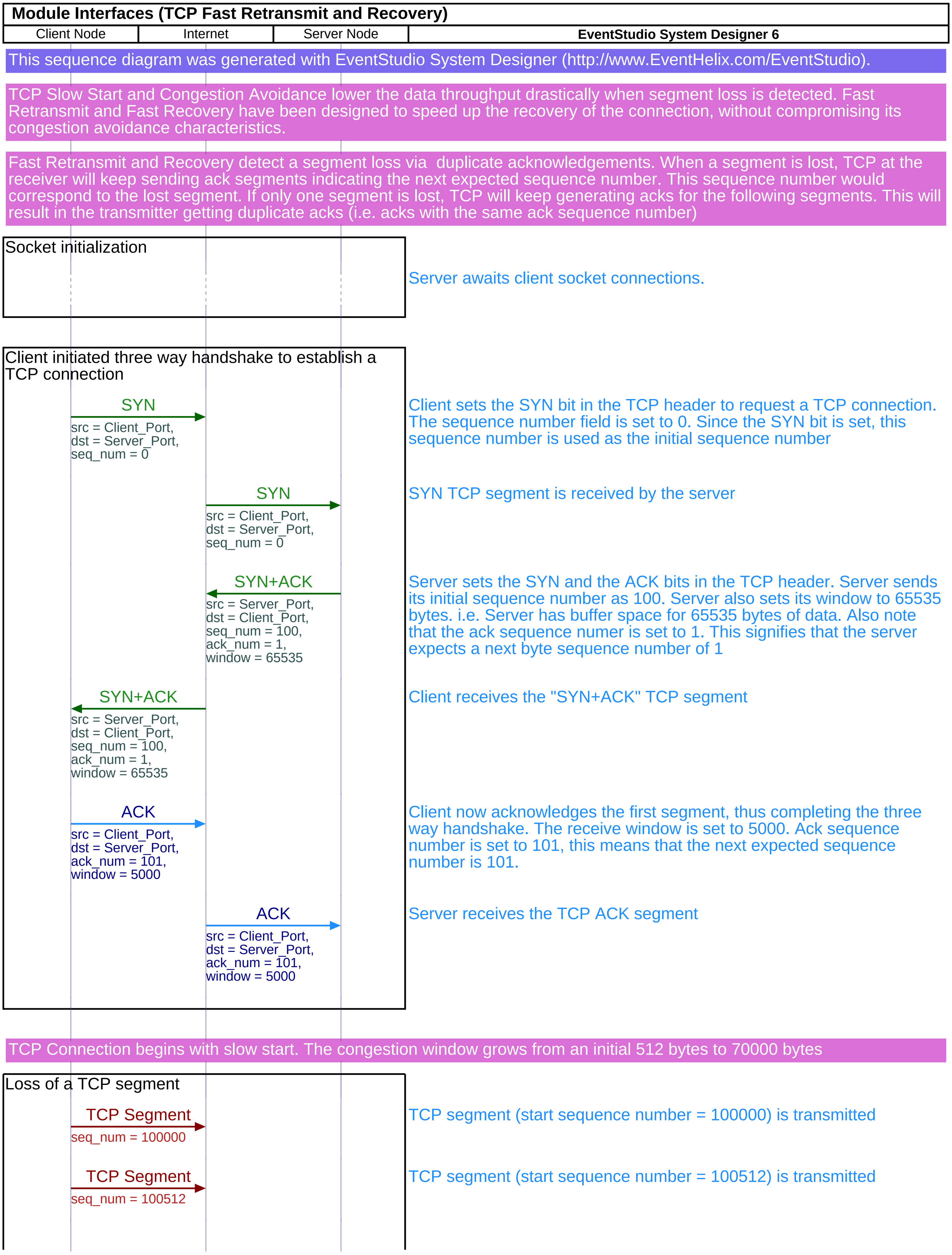
Module Interfaces (TCP Fast Retransmit and Recovery)



Client Node	Internet	Server Node	EventStudio System Designer 6

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 awaits client socket connections.

Client initiated three way handshake to establish a TCP connection

SYN
src = Client_Port,
dst = Server_Port,
seq_num = 0

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

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

Client receives the "SYN+ACK" TCP segment

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

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.

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

Server receives the TCP ACK segment

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
seq_num = 100000

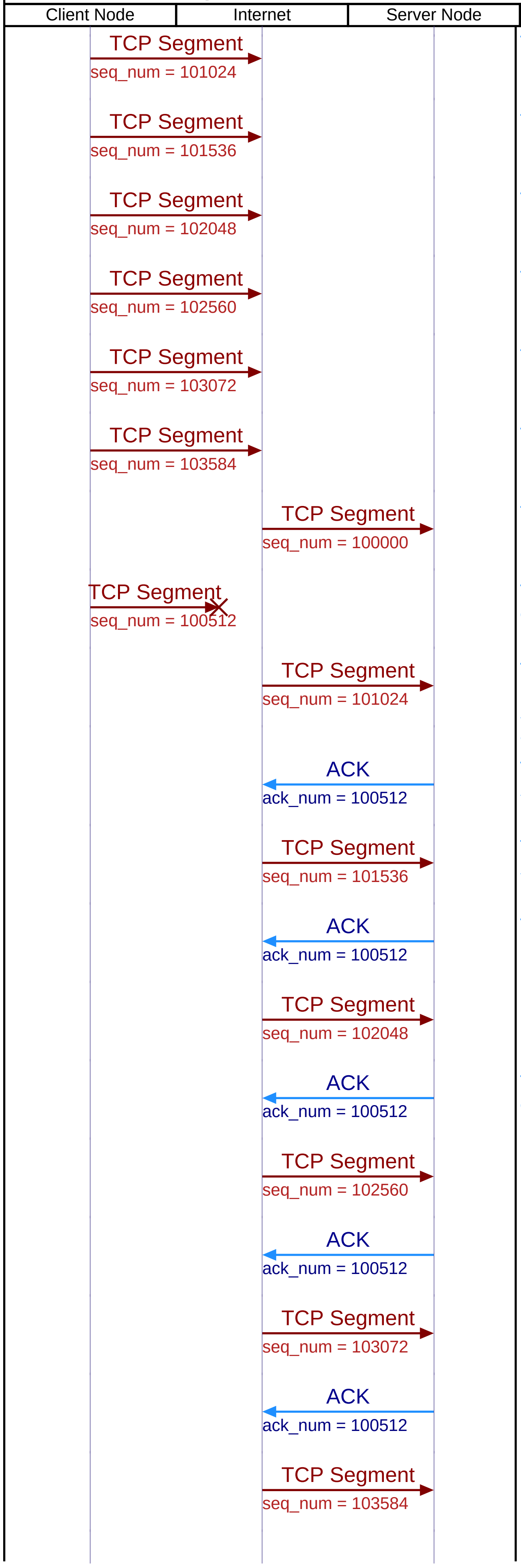
TCP segment (start sequence number = 100000) is transmitted

TCP Segment
seq_num = 100512

TCP segment (start sequence number = 100512) is transmitted

Module Interfaces (TCP Fast Retransmit and Recovery)

EventStudio System Designer 6



TCP segment (start sequence number = 101024) is transmitted

TCP segment (start sequence number = 101536) is transmitted

TCP segment (start sequence number = 102048) is transmitted

TCP segment (start sequence number = 102560) is transmitted

TCP segment (start sequence number = 103072) is transmitted

TCP segment (start sequence number = 103584) is transmitted

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

TCP segment (start sequence number = 100512) is lost due to congestion in the network.

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.

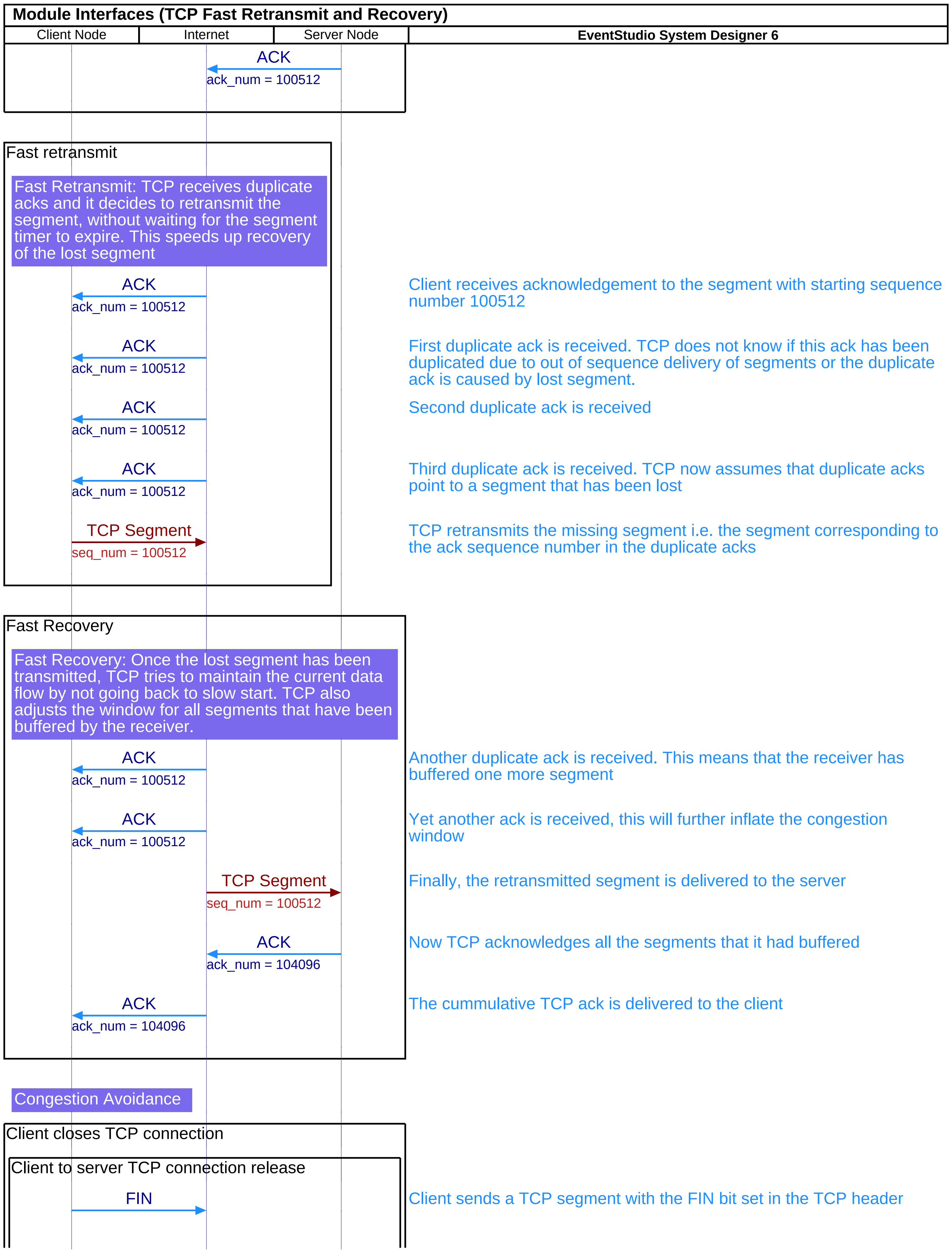
TCP sends an acknowledgement to the Sender with the next expected sequence number set to 100512.

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

Module Interfaces (TCP Fast Retransmit and Recovery)



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

Client receives acknowledgement to the segment with starting sequence number 100512

First duplicate ack is received. TCP does not know if this ack has been duplicated due to out of sequence delivery of segments or the duplicate ack is caused by lost segment.

Second duplicate ack is received

Third duplicate ack is received. TCP now assumes that duplicate acks point to a segment that has been lost

TCP retransmits the missing segment i.e. the segment corresponding to the ack sequence number in the duplicate acks

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.

Another duplicate ack is received. This means that the receiver has buffered one more segment

Yet another ack is received, this will further inflate the congestion window

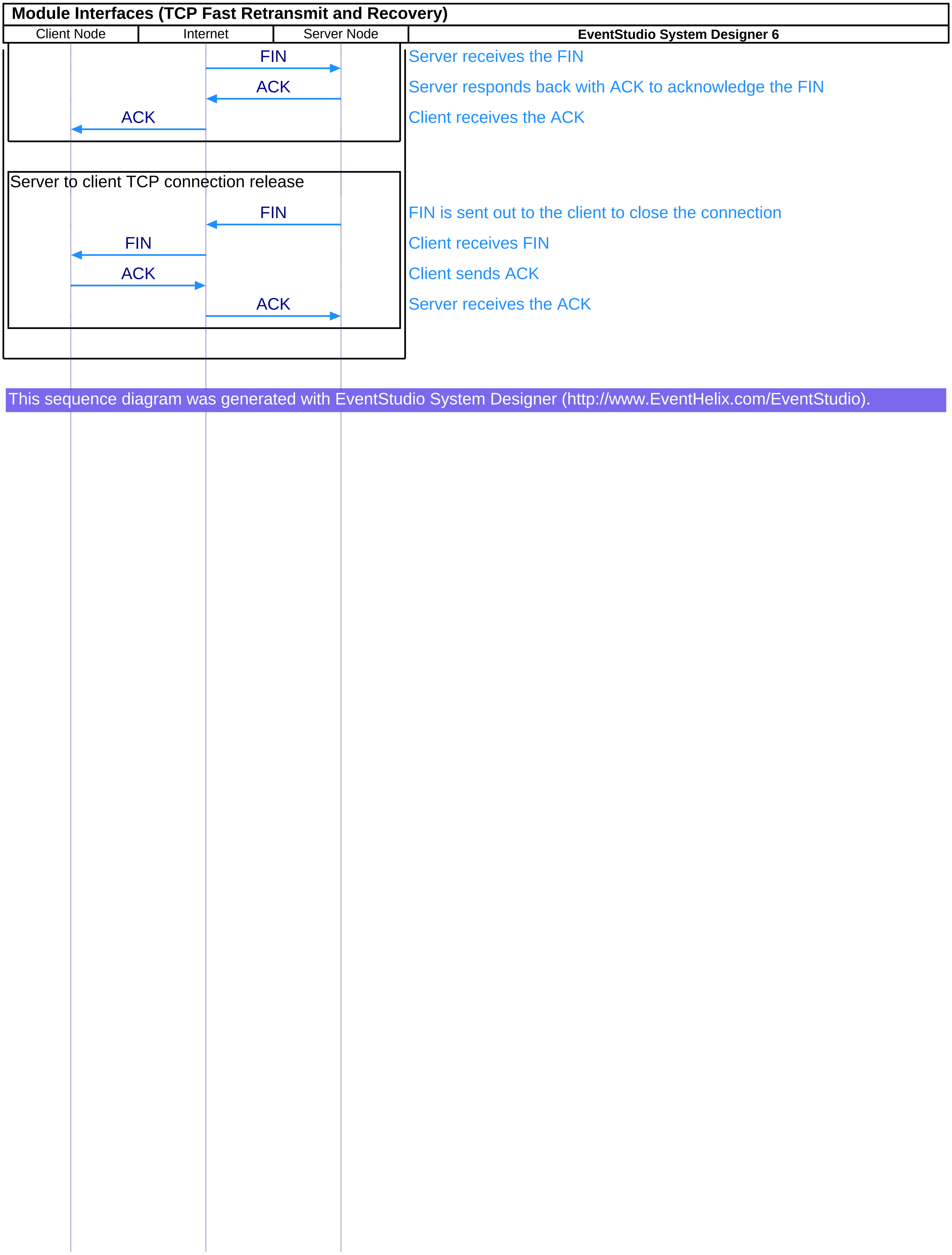
Finally, the retransmitted segment is delivered to the server

Now TCP acknowledges all the segments that it had buffered

The cummulative TCP ack is delivered to the client

Congestion Avoidance

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


Module Interfaces (TCP Fast Retransmit and Recovery)
Client Node
Internet
Server Node
EventStudio System Designer 6
FIN
ACK
ACK
Server to client TCP connection release
FIN
FIN
ACK
ACK

Server receives the FIN

Server responds back with ACK to acknowledge the FIN

Client receives the ACK

FIN is sent out to the client to close the connection

Client receives FIN

Client sends ACK

Server receives the ACK