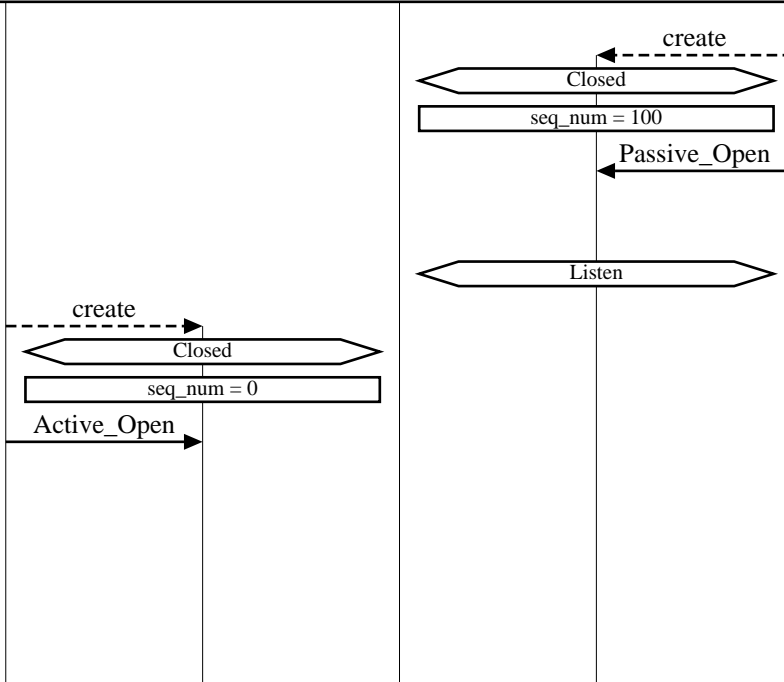


TCP - Transmission Control Protocol (TCP Basic Handling)					
Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 1)

Copyright © 2000-2003 EventHelix.com Inc. All Rights Reserved.

LEG: About TCP

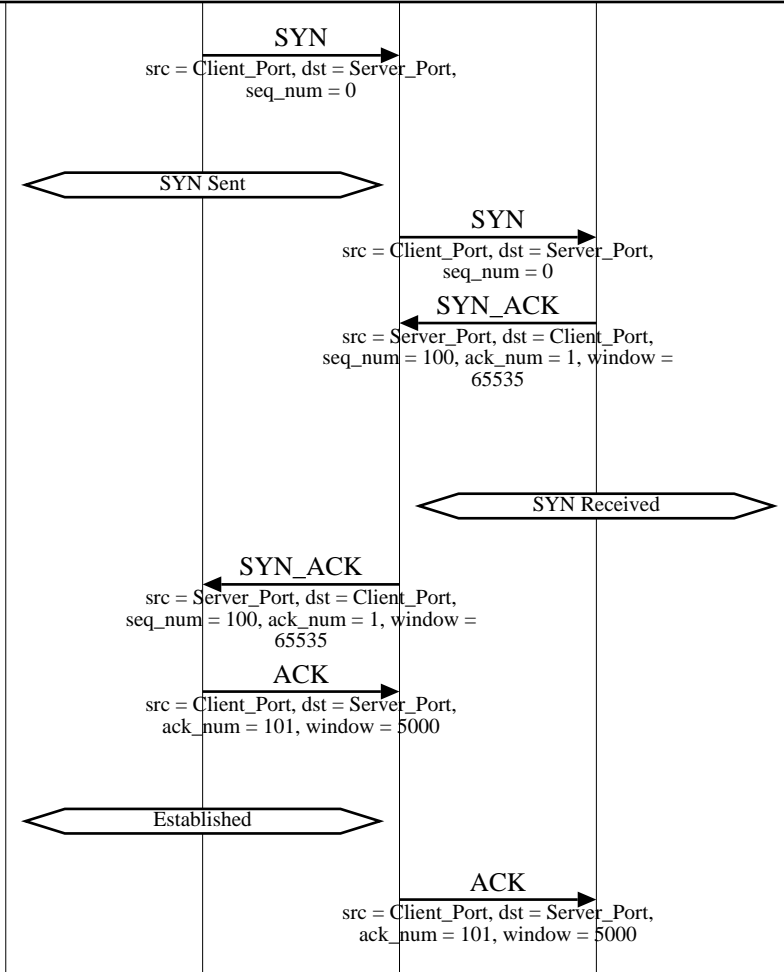
TCP (Transmission Control Protocol) provides a reliable end to end service that delivers packets over the Internet. Packets are delivered in sequence without loss or duplication.



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

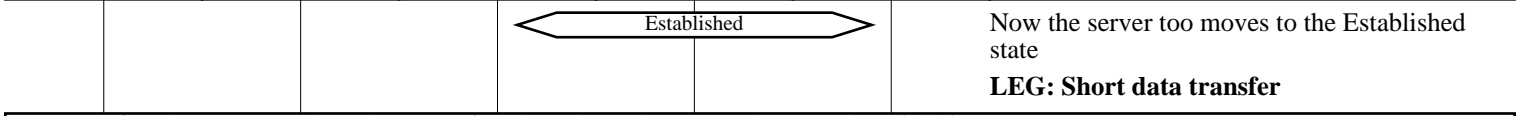
Client initiated three way handshake to establish a TCP connection



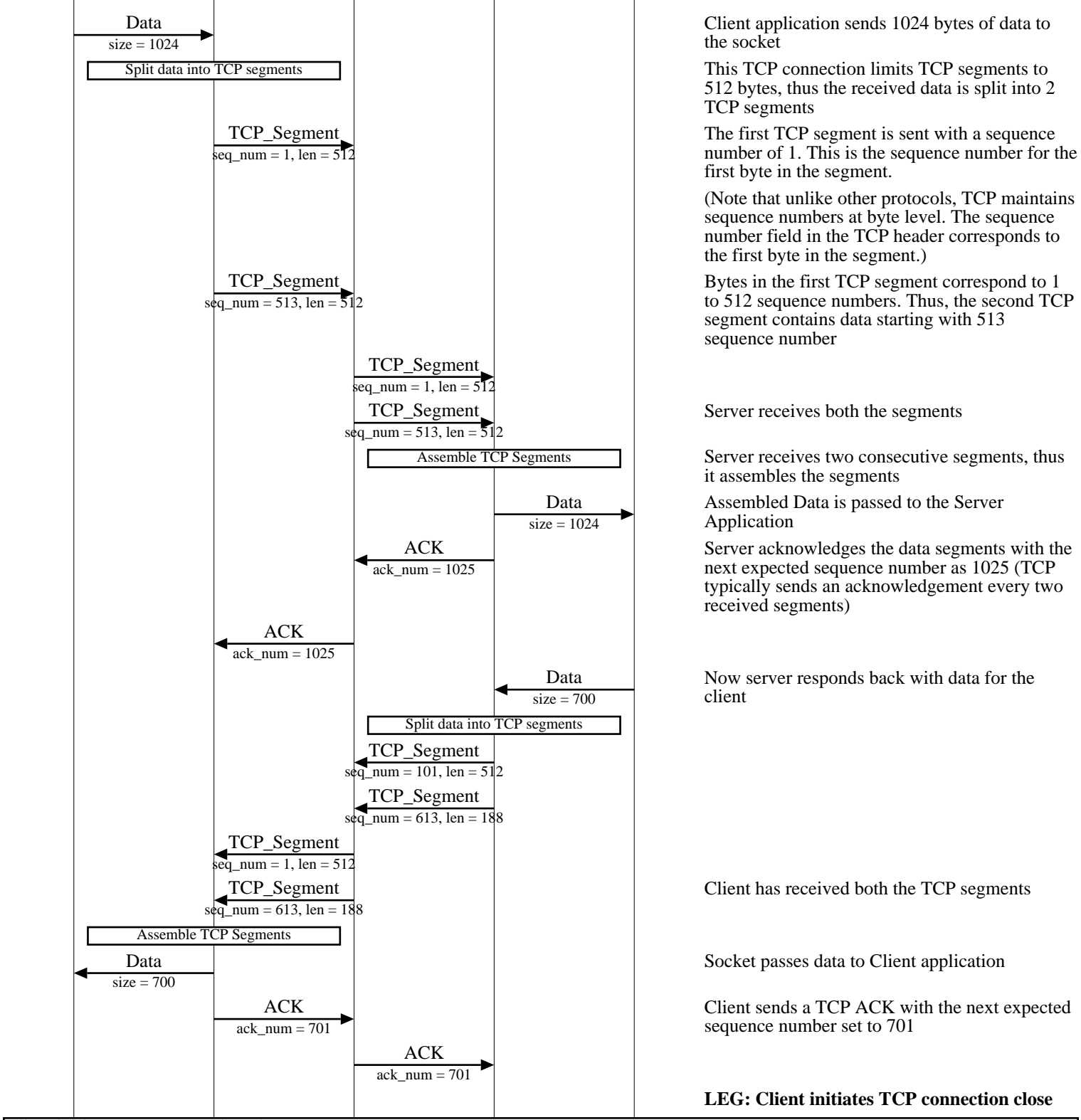
Client sets the SYN bit in the TCP header to request a TCP connection. The sequence number field is set to 0. Since the SYN bit is set, this sequence number is used as the initial sequence number
 Socket transitions to the SYN Sent state
 SYN TCP segment is received by the server
 Server sets the SYN 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 number 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

TCP - Transmission Control Protocol (TCP Basic Handling)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 2)



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

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 701

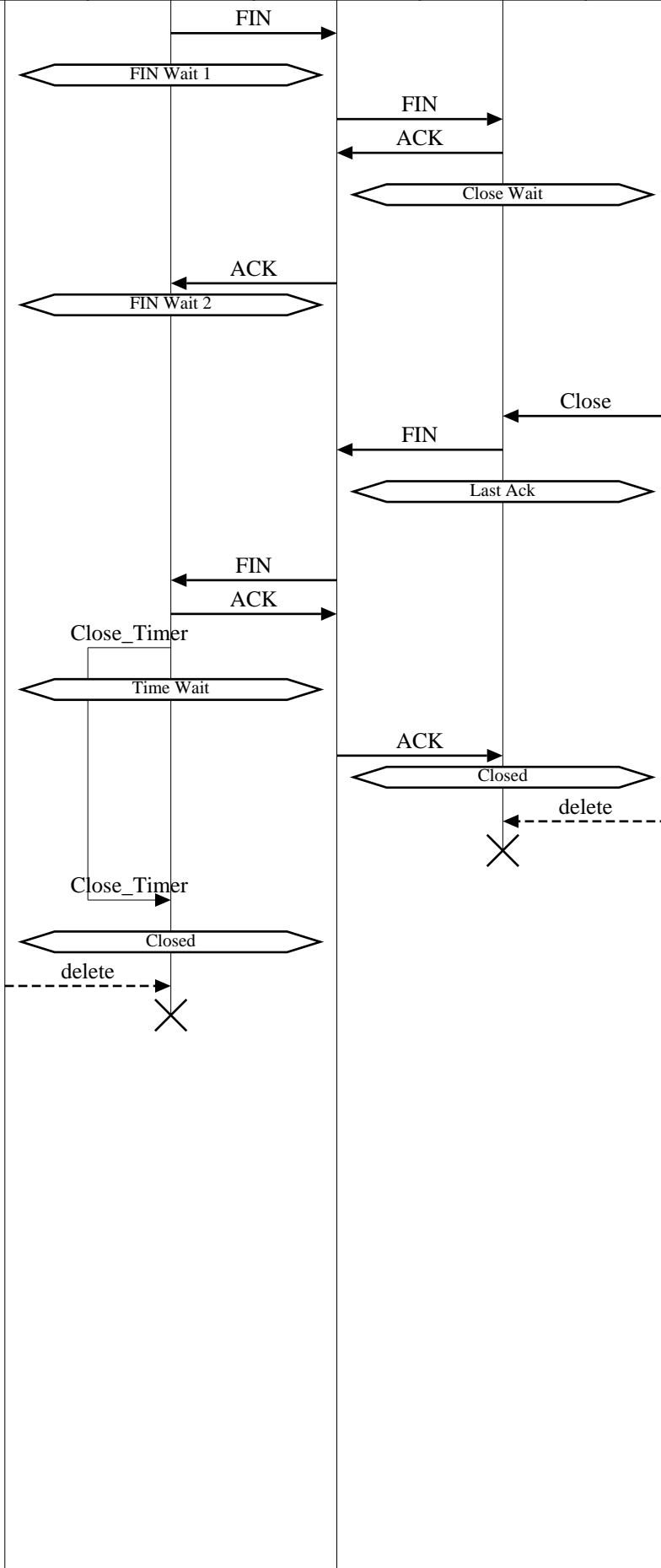
LEG: Client initiates TCP connection close

Client initiates TCP connection close



TCP - Transmission Control Protocol (TCP Basic Handling)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 3)



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

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.

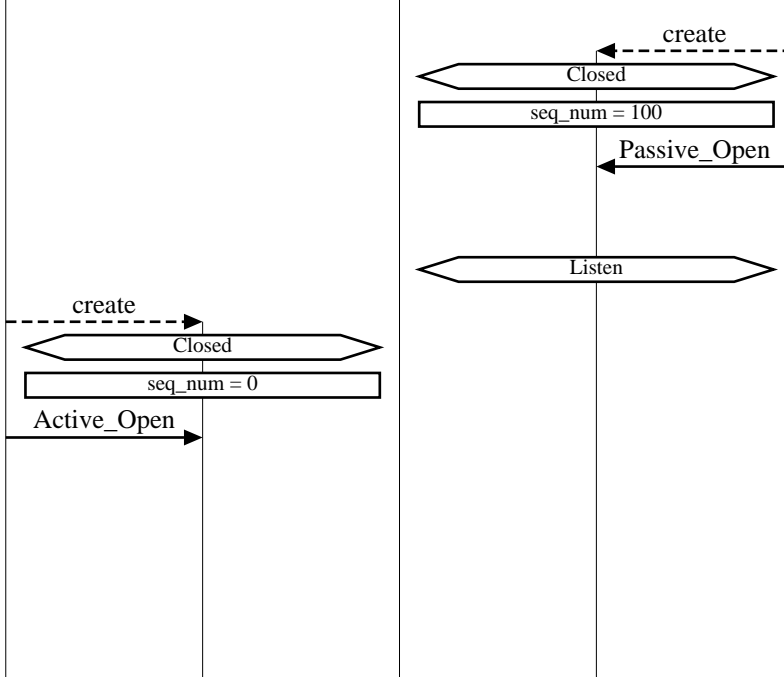
TCP - Transmission Control Protocol (TCP Slow Start)					
Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 4)

Copyright © 2000-2003 EventHelix.com Inc. All Rights Reserved.

LEG: About TCP Slow Start

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.

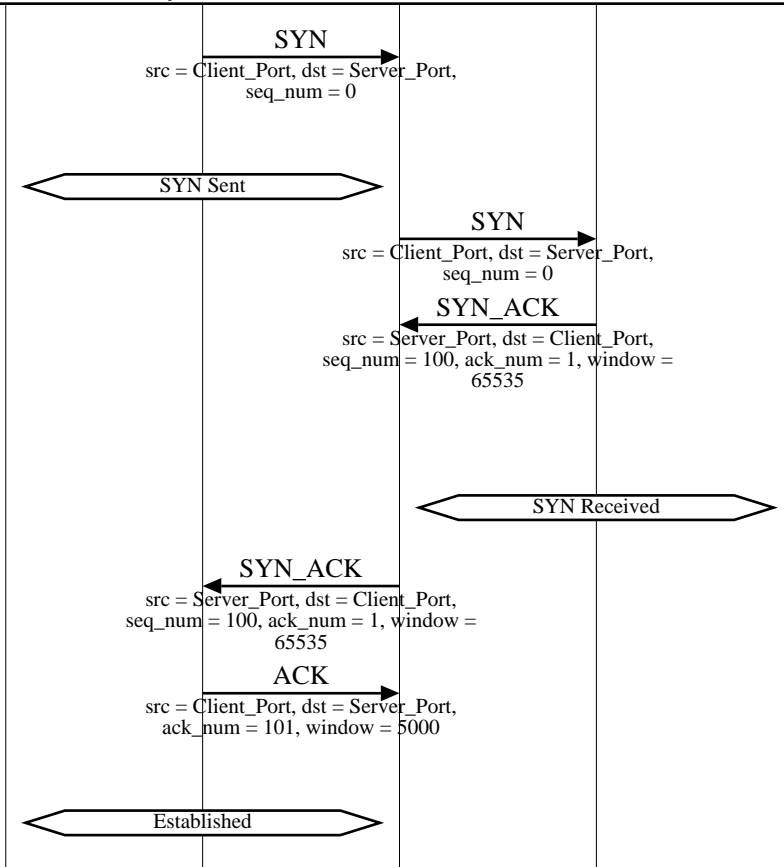
In this sequence diagram we will analyse "Slow start", an important part of the congestion control mechanisms built right into TCP. As the name suggests, "Slow Start" starts slowly, increasing its window size as it gains confidence about the networks throughput.



Server 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

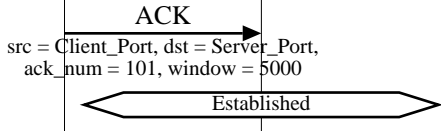
Client initiated three way handshake to establish a TCP connection



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 number 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

TCP - Transmission Control Protocol (TCP Slow Start)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 5)



Server receives the TCP ACK segment

Now the server too moves to the Established state

LEG: TCP Slow Start

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 networks 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

cwnd = 512 (1 segment)

ssthresh = 65535

Slow Start

cwnd = 512 (1 segment)

ssthresh = 65535

Slow Start

Data
size = 5120

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

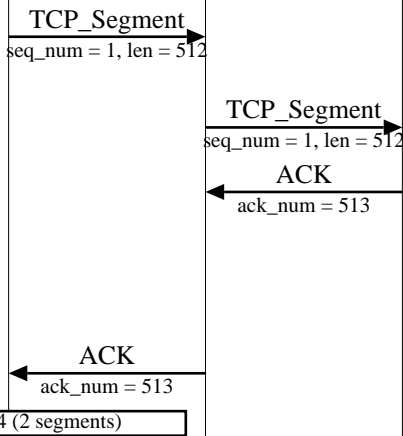
Client end TCP connection moves to slow start state

By the same logic, the server also sets cwnd to 512

Server end TCP connection moves to slow start state

Client application sends 5120 bytes of data to the socket

Roundtrip #1 of data transmission



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

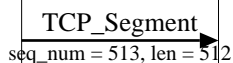
Server acknowledges the data segments with the next expected sequence number as 513

TCP typically sends an acknowledgement every two received segments but in this case it times out for another segment and decides to acknowledge the only segment received.

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.

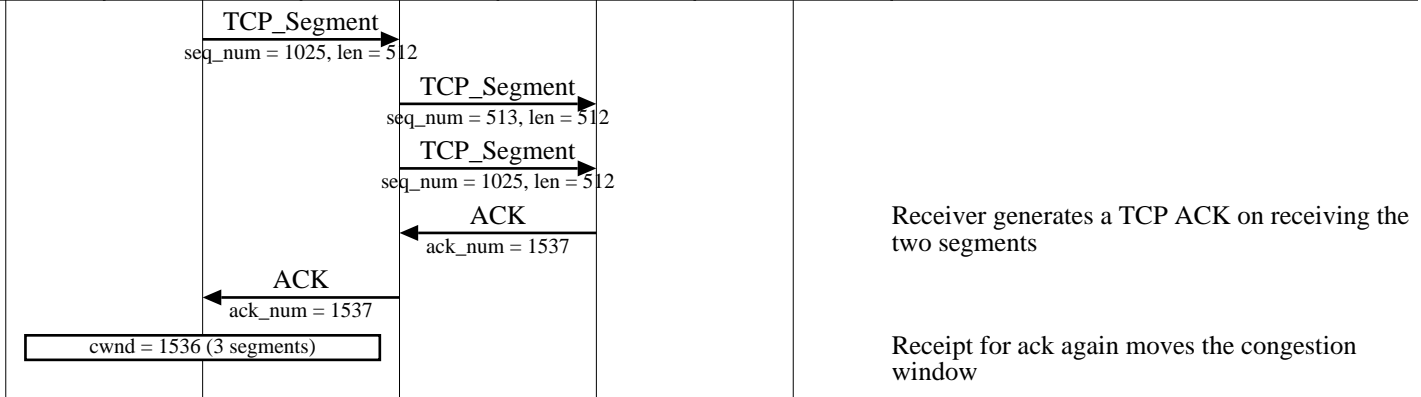
Roundtrip #2 of data transmission



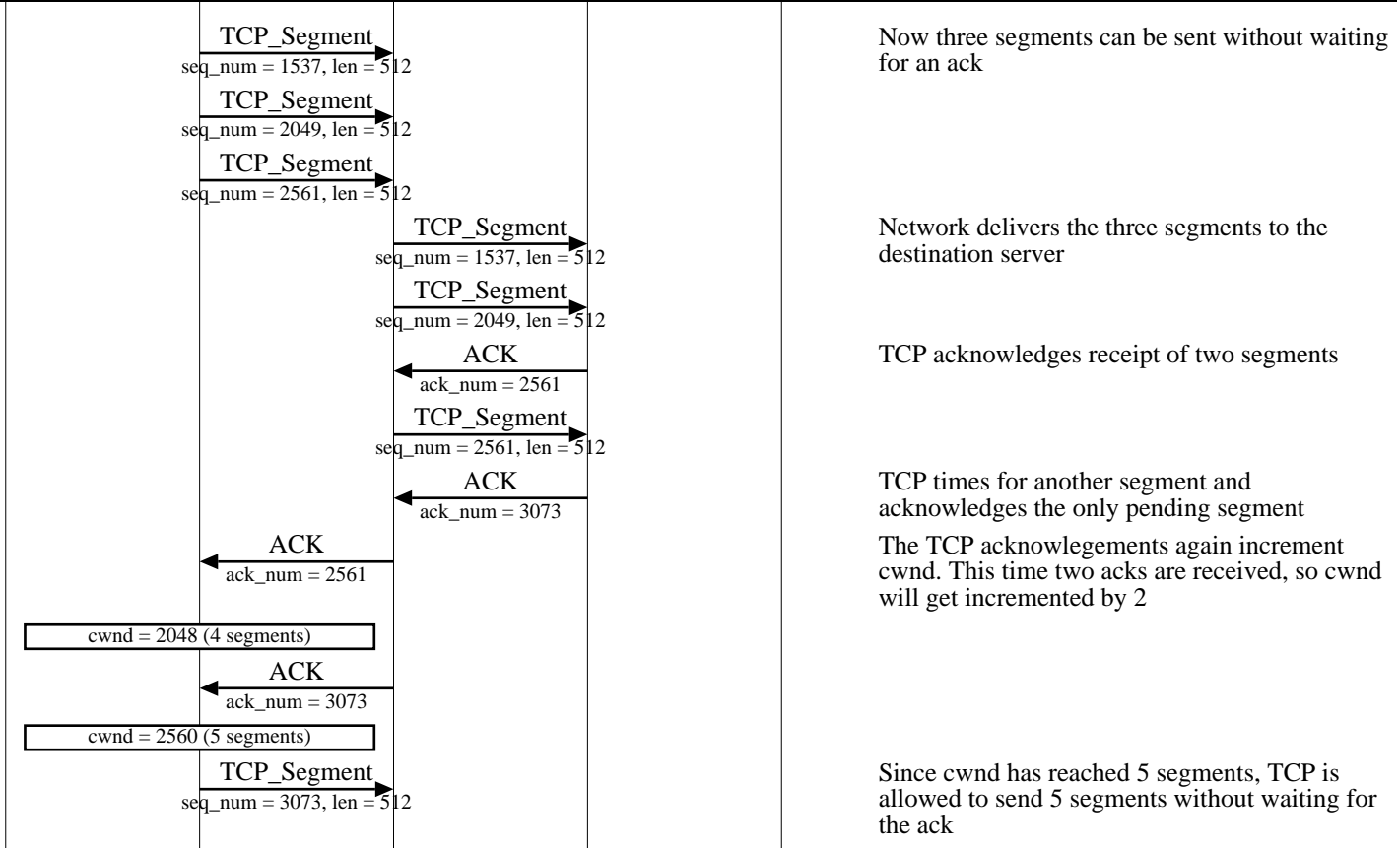
Since the congestion window has increased to 2, TCP can now send two segments without waiting for an ack

TCP - Transmission Control Protocol (TCP Slow Start)

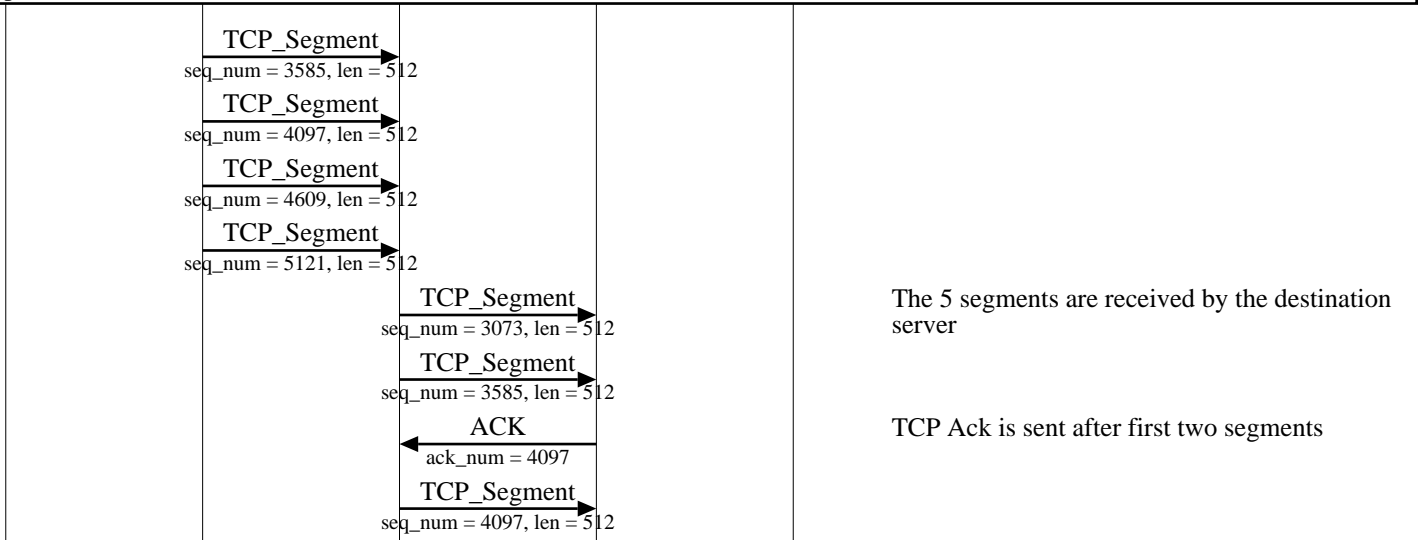
Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 6)



Roundtrip #3 of data transmission



Roundtrip #4 of data transmission

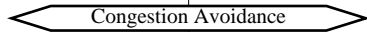


TCP - Transmission Control Protocol (TCP Slow Start)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 8)

Within a few more roundtrip interactions cwnd will exceed ssthresh. At this point the session will be considered out of slow start. Note that the TCP connection from the client side is out of slow start but the server end is still in slow start as it has not sent any data to the client.

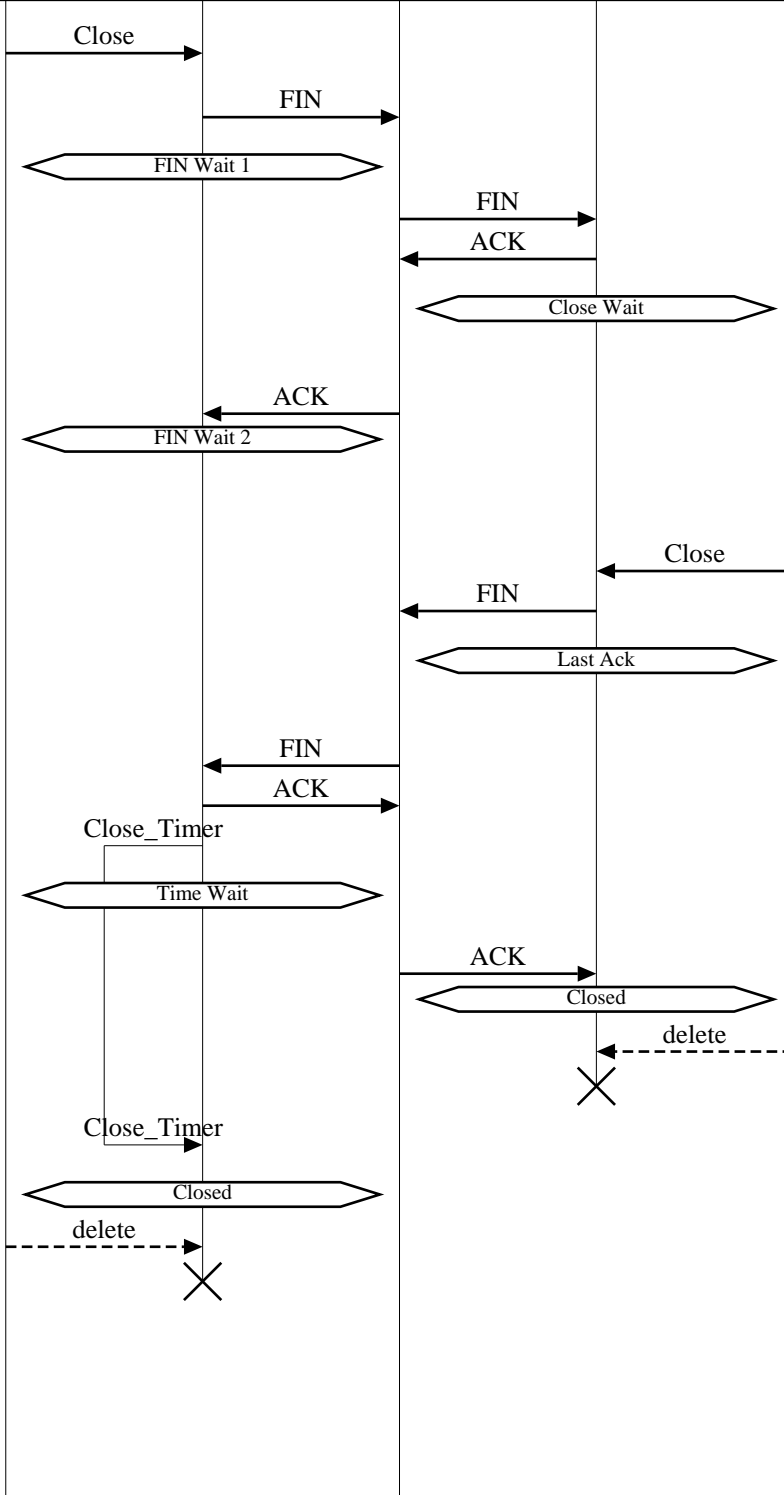
Exiting slow start signifies that the TCP connection has reached an equilibrium state where the congestion window closely matches the networks capacity. From this point on, the congestion window will not move geometrically. cwnd will move linearly once the connection is out of slow start.



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

LEG: Client initiates TCP connection close

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

Client waits in Time Wait state to handle a FIN retry

Server receives the ACK

Server moves the connection to closed state

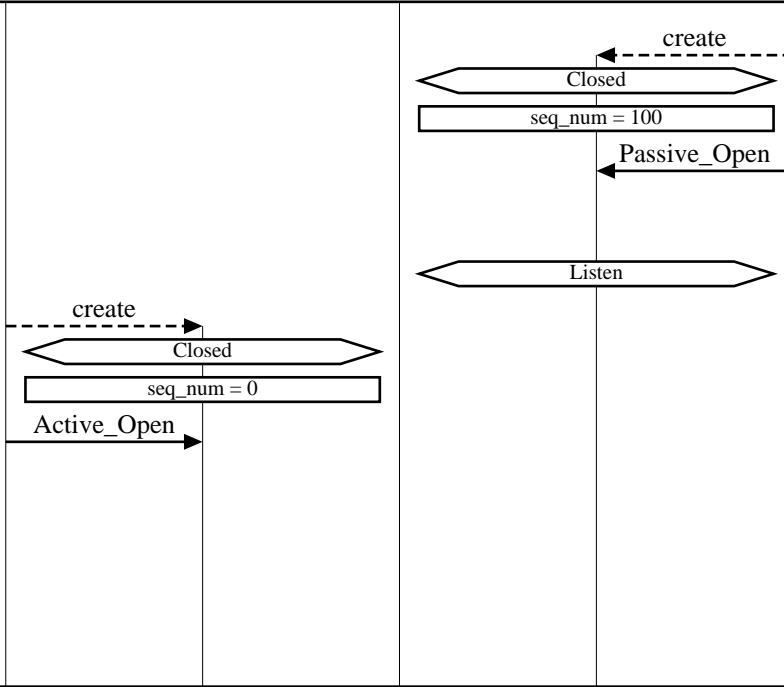
Close timer has expired. Thus the client end connection can be closed too.

TCP - Transmission Control Protocol (TCP Congestion Avoidance)					
Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 9)

Copyright © 2000-2003 EventHelix.com Inc. All Rights Reserved.

LEG: About TCP Congestion Avoidance

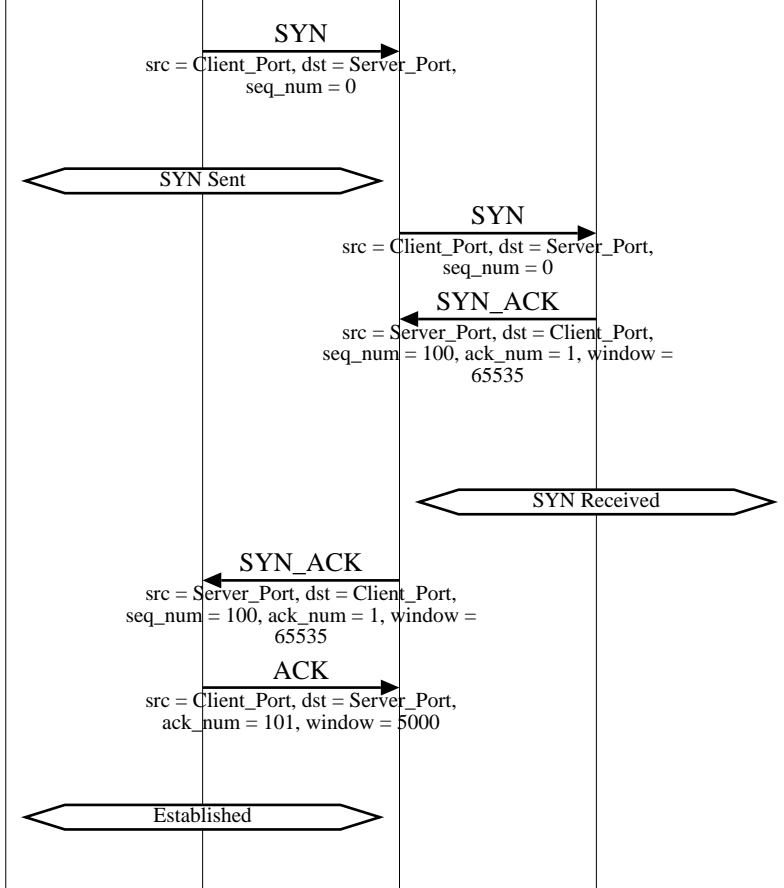
We have already seen that TCP connection starts up in slow start mode, geometrically increasing the congestion window (cwnd) until it crosses the slow start threshold (ssthresh). Once cwnd is greater than ssthresh, TCP enters the congestion avoidance mode of operation. In this mode, the primary objective is to maintain high throughput without causing congestion. If TCP detects segment loss, it assumes that congestion has been detected over the internet. As a corrective action, TCP reduces its data flow rate by reducing cwnd. After reducing cwnd, TCP goes back to slow start.



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

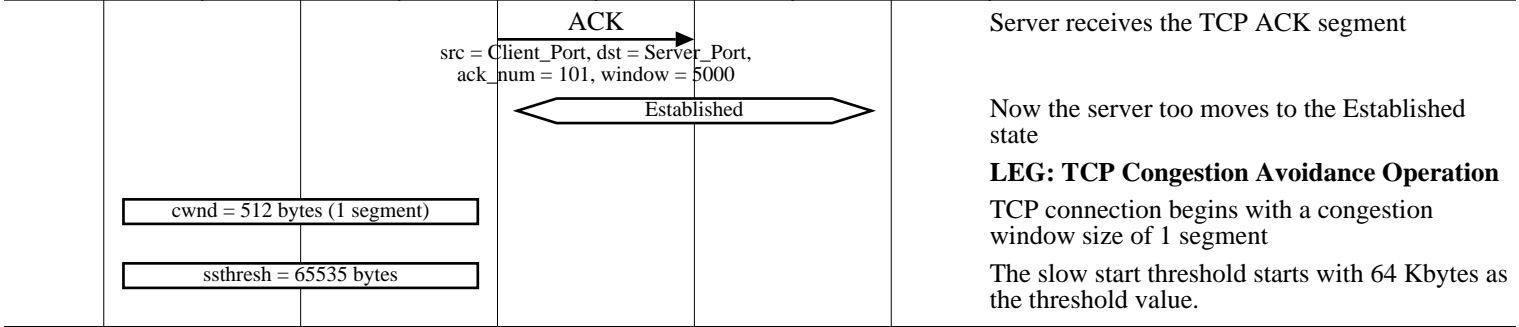
Client initiated three way handshake to establish a TCP connection



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 number 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

TCP - Transmission Control Protocol (TCP Congestion Avoidance)

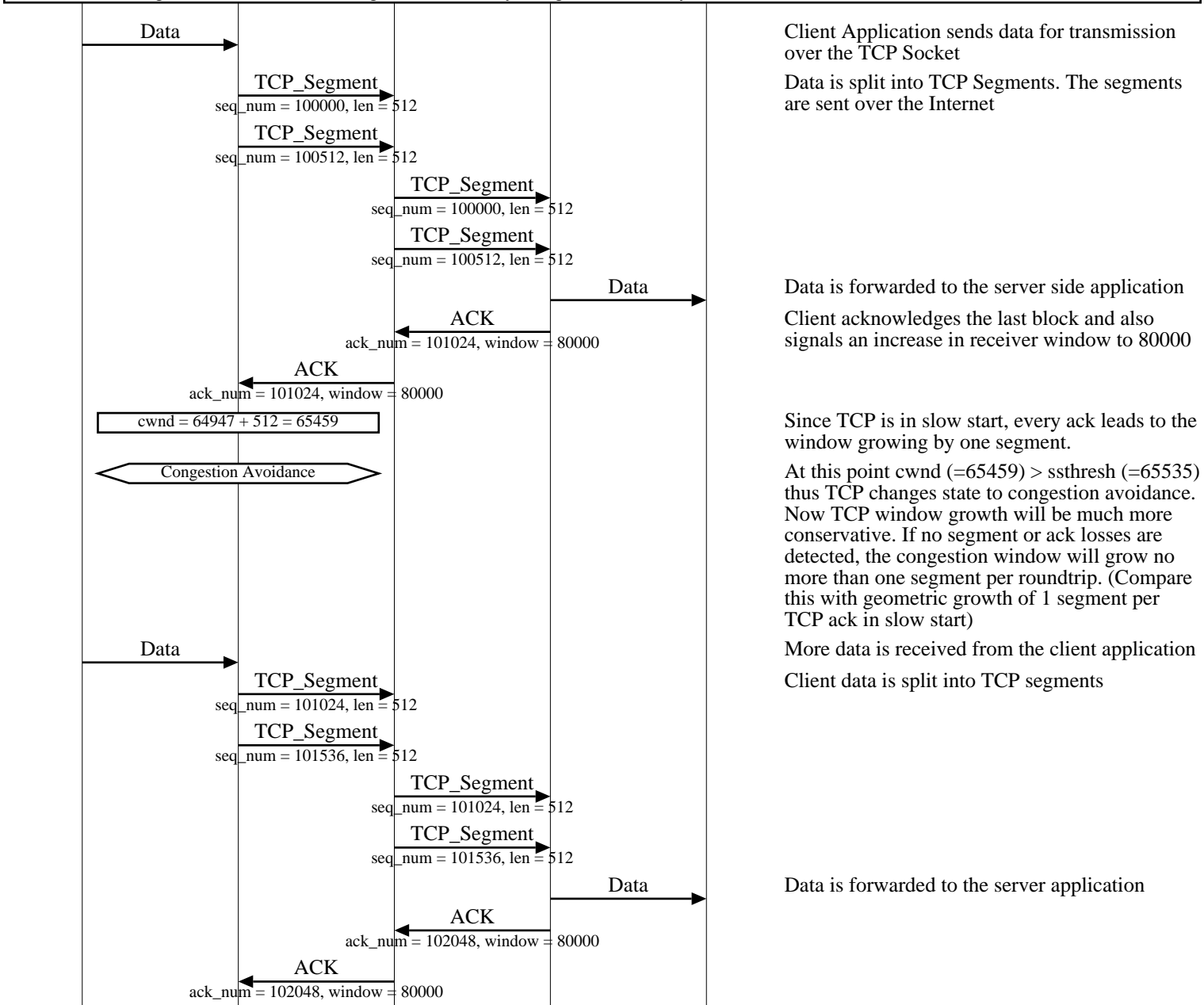
Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 10)



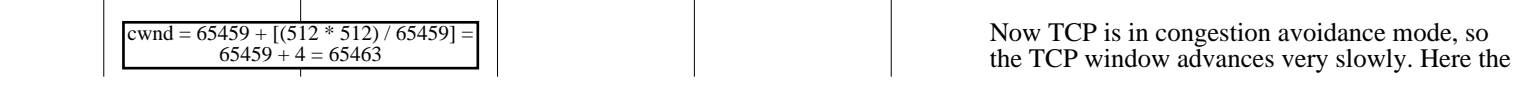
TCP session begins with "Slow Start". See the sequence diagram on slow start for details



TCP congestion window grows from 512 bytes (1 segment) to 64947 (assuming no segment losses are detected during slow start). During slow start the congestion window was being incremented by 1 segment for every TCP Ack from the other end.

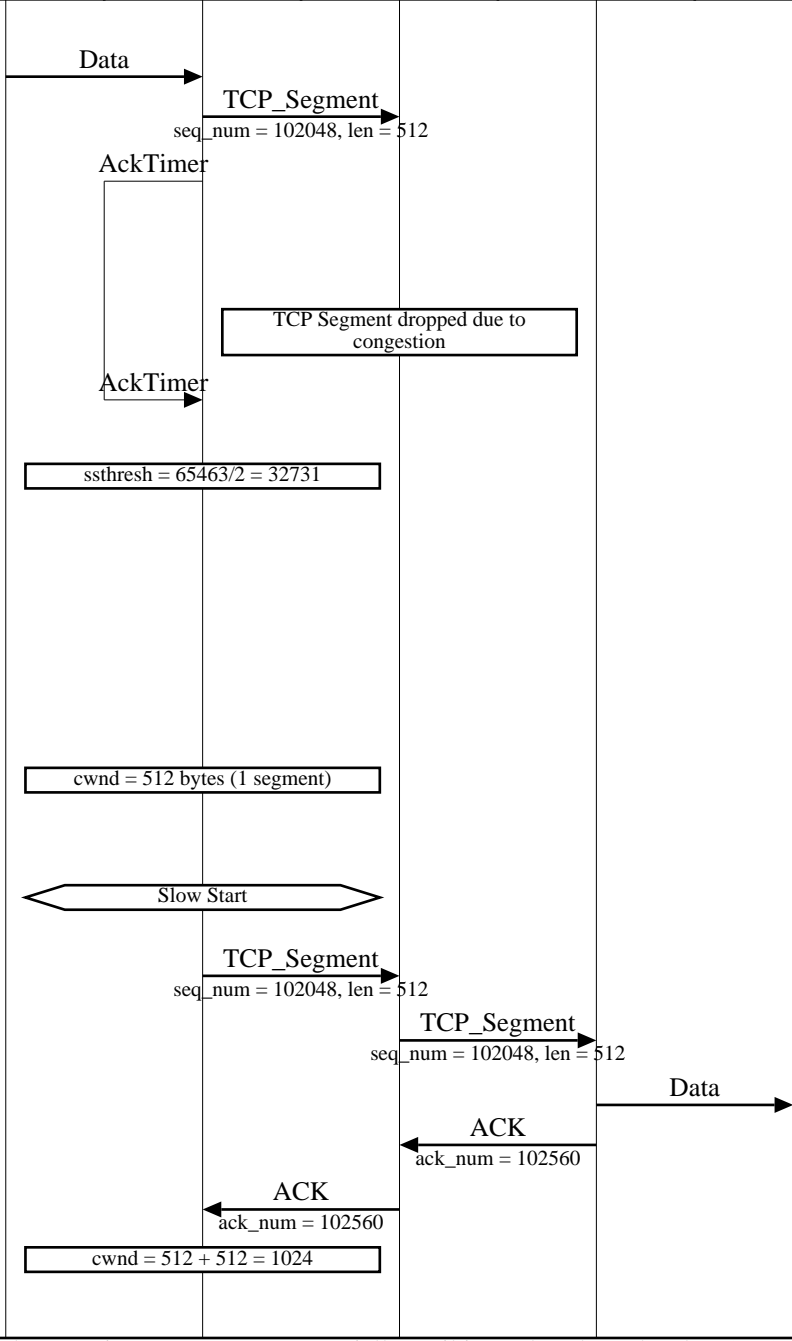


$cwnd$ is incremented using the formula: $cwnd = cwnd + (segment_size \cdot segment_size) / cwnd$



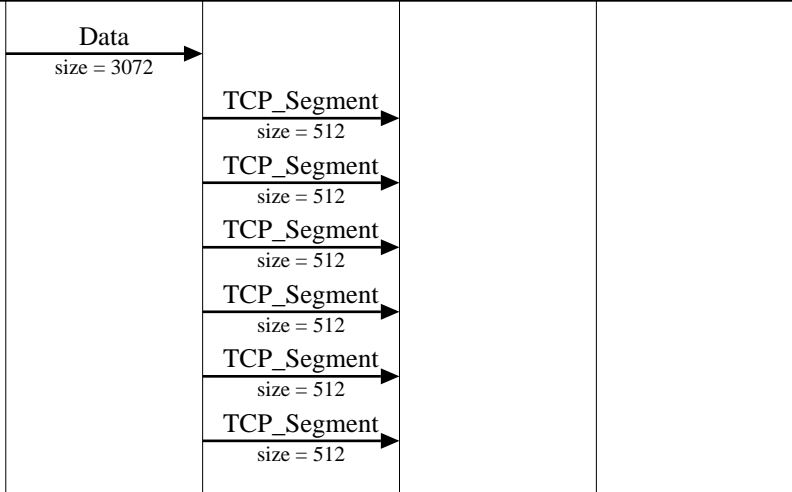
TCP - Transmission Control Protocol (TCP Congestion Avoidance)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 11)



window increased by only 4 bytes.
 Data to be sent to server
 TCP session sends out the data as a single segment
 TCP session starts a ack timer, awaiting the TCP ack for this segment.
 Note: The above timer is started for every segment. The timer is not shown at other places as it played role in our analysis
 Some node in the Internet drops the TCP segment due to congestion
 TCP times out for a TCP ACK from the other end. This will be treated as a sign of congestion by TCP
 When TCP detects congestion, it stores half of the current congestion window in ssthresh variable. In this case, ssthresh has been reduced from 65535 to 32731. This signifies that TCP now has less confidence on the network's ability to support big window sizes. Thus if the window size falls due to congestion, rapid window size increases will be carried out only until the window reaches 32731. Once this lowered ssthresh value is reached, window growth will be much slower.
 Since current congestion has been detected by timeout, TCP takes the drastic action of reducing the congestion window to 1. As you can see, this will have a big impact on the throughput.
 cwnd (=1) is now lower than ssthresh (=32731) so TCP goes back to slow start.
 Data is finally given to the server application
 Since TCP is in slow start, a TCP acknowledgement results in the window growing by one segment

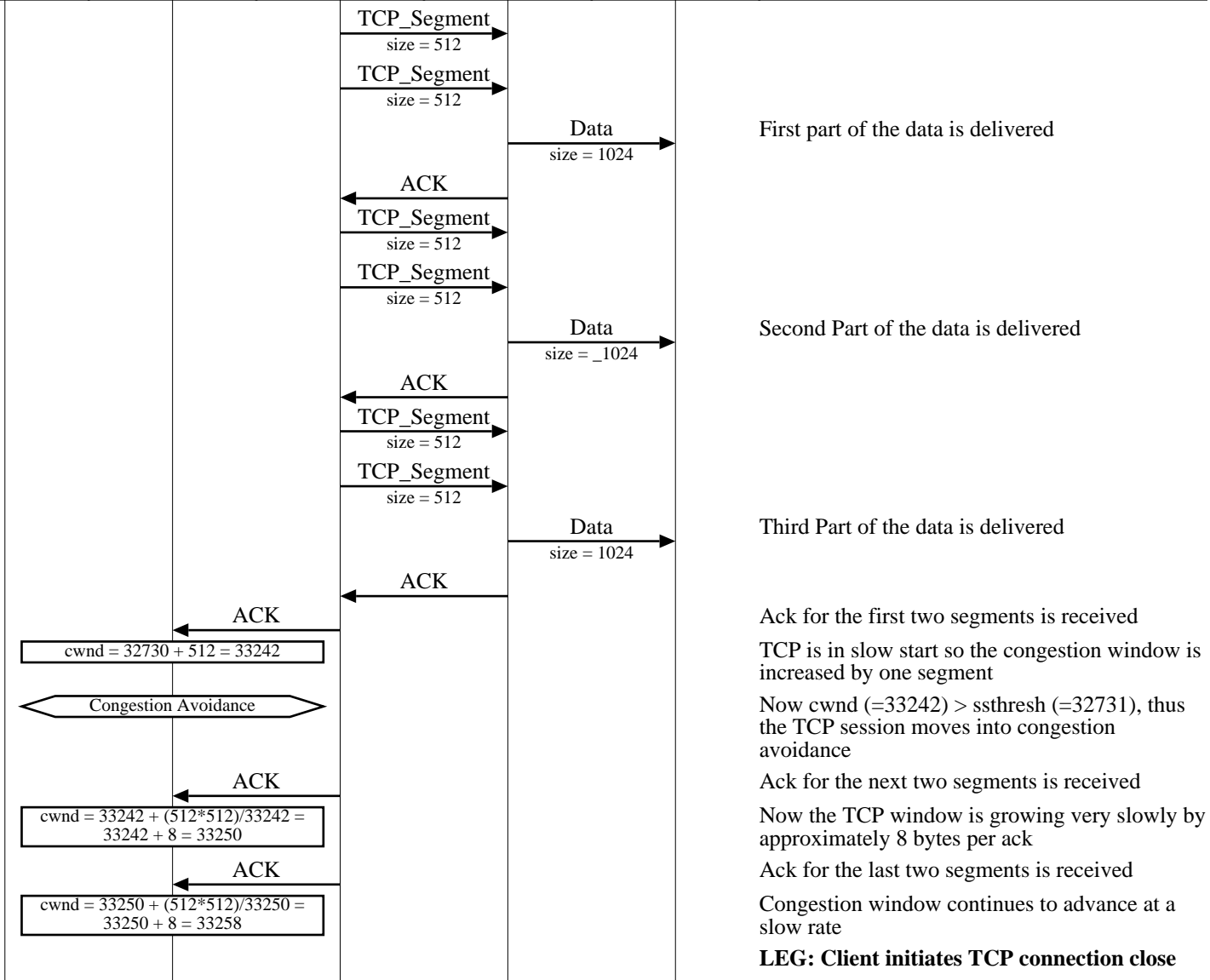
TCP window continues to grow exponentially until it reaches the ssthresh (=32731) value.



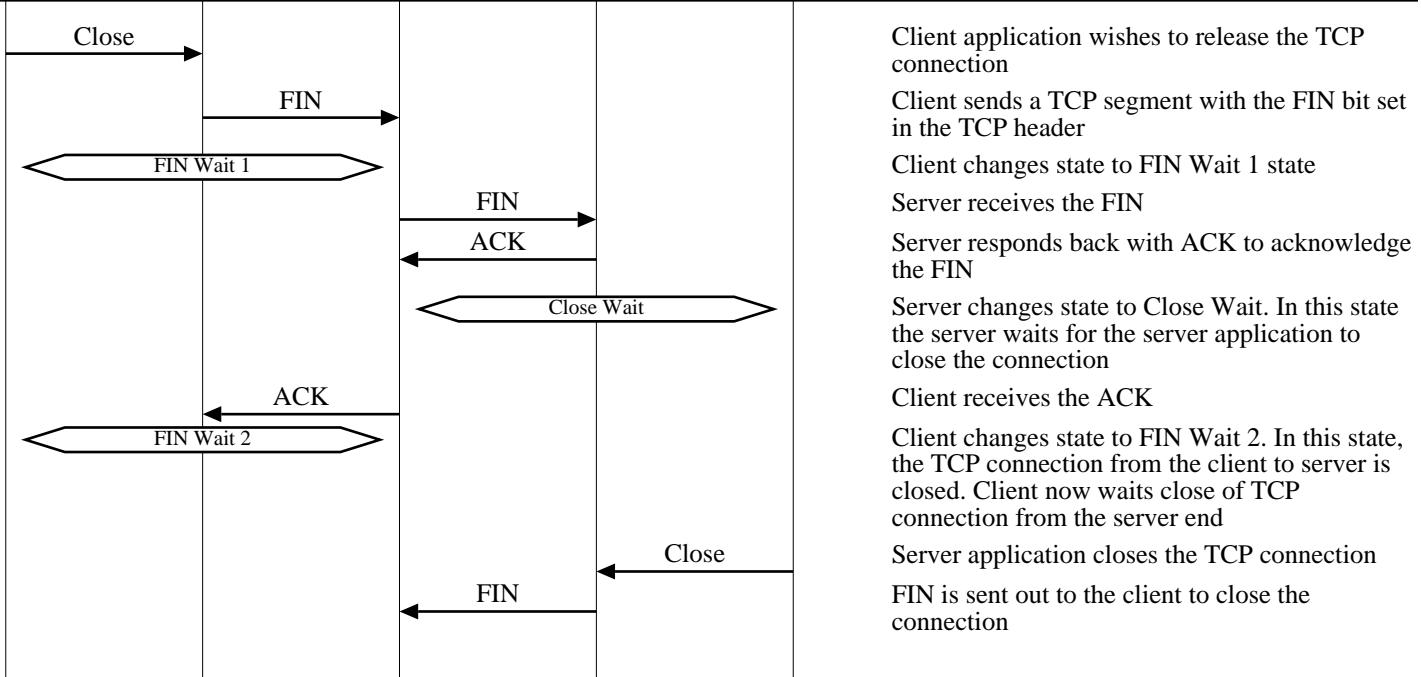
Six TCP segments are transmitted in the slow start mode

TCP - Transmission Control Protocol (TCP Congestion Avoidance)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 12)

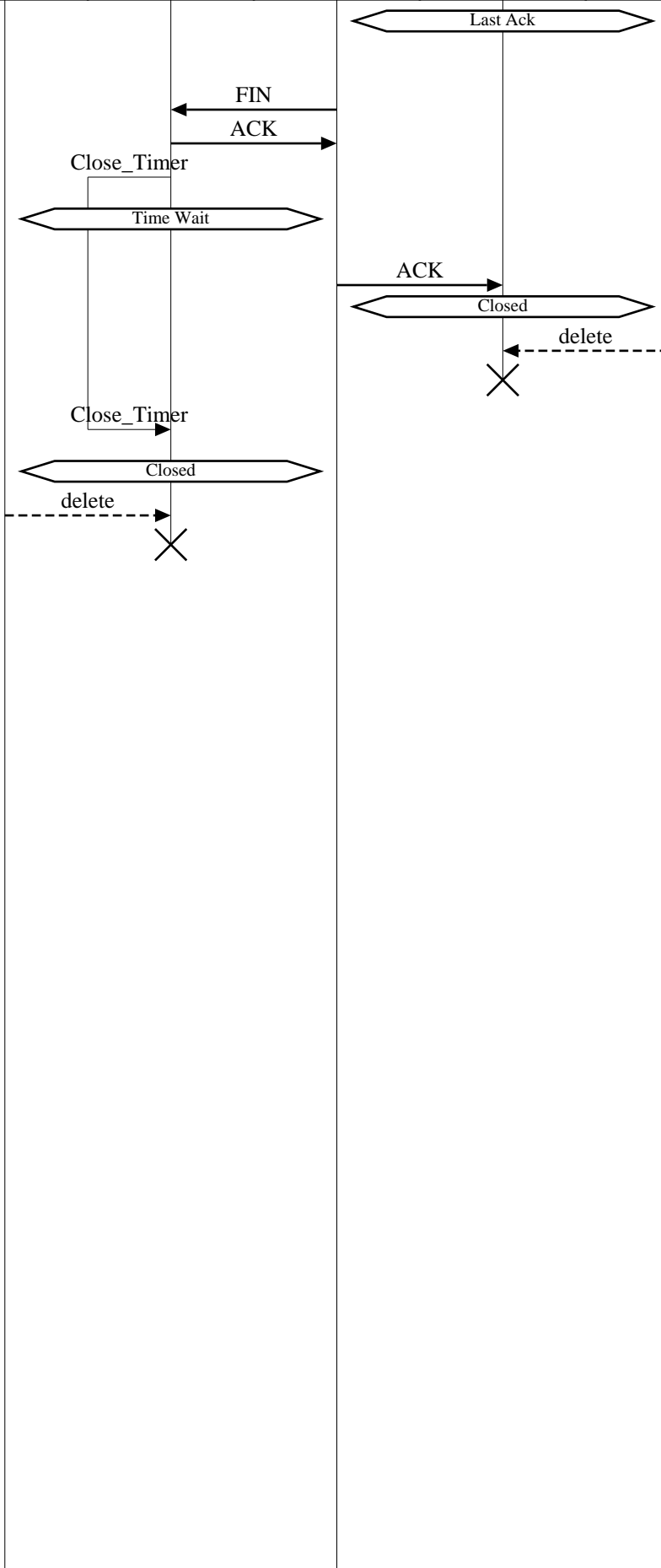


Client initiates TCP connection close



TCP - Transmission Control Protocol (TCP Congestion Avoidance)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 13)



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

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.

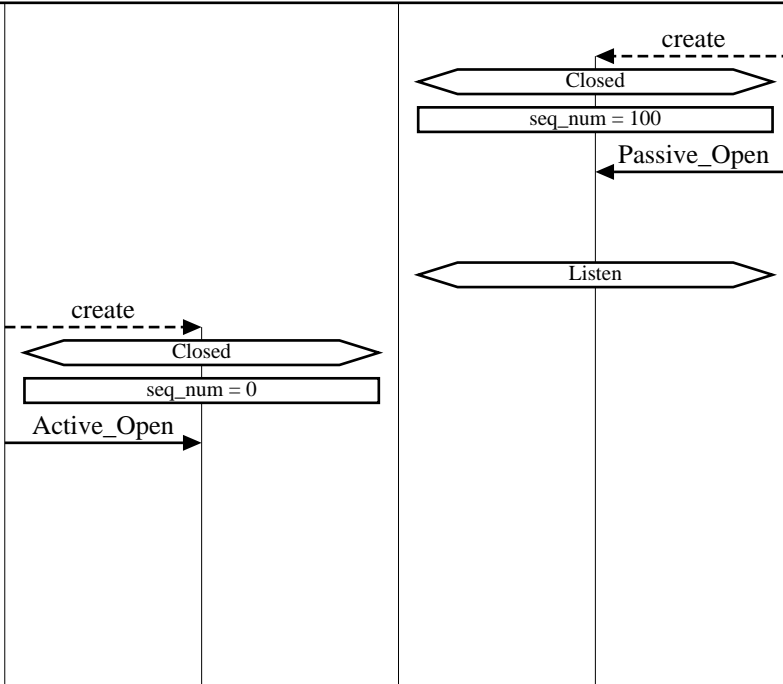
TCP - Transmission Control Protocol (TCP Fast Transmit and Recovery)					
Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 14)

Copyright © 2000-2003 EventHelix.com Inc. All Rights Reserved.

LEG: About Fast Retransmit and Fast Recovery

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.

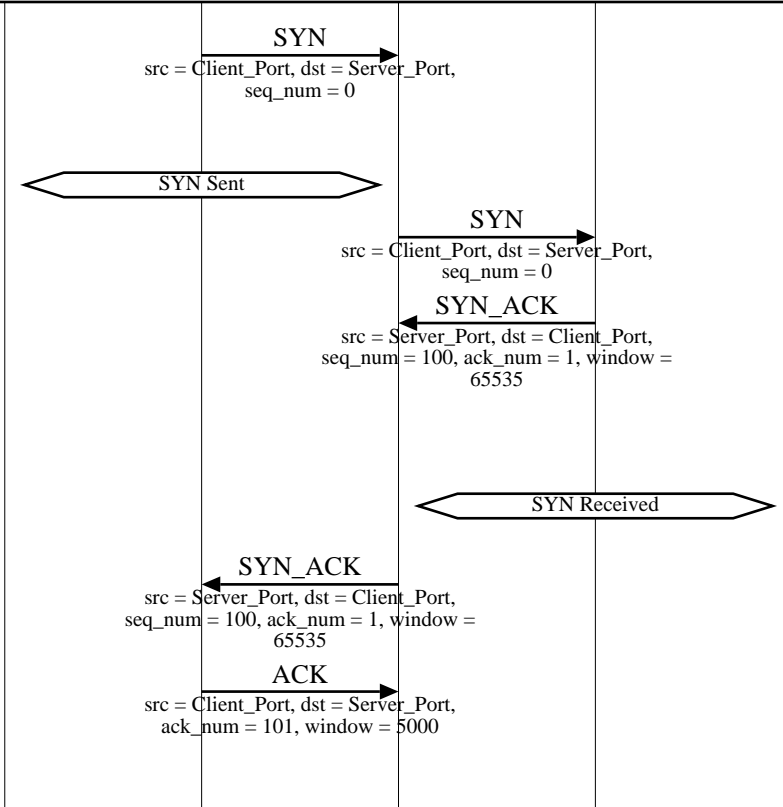
Fast Retransmit and Recovery detect a segment loss via duplicate acknowledgements. When a segment is lost, TCP at the receiver will keep sending ack segments indicating the next expected sequence number. This sequence number would correspond to the lost segment. If only one segment is lost, TCP will 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)



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

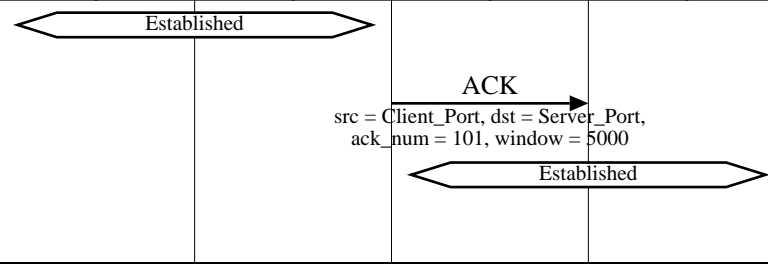


Client sets the SYN bit in the TCP header to request a TCP connection. The sequence number field is set to 0. Since the SYN bit is set, this sequence number is used as the initial sequence number
 Socket transitions to the SYN Sent state
 SYN TCP segment is received by the server
 Server sets the SYN 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 number 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.

TCP - Transmission Control Protocol (TCP Fast Transmit and Recovery)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	

04-Jul-03 14:31 (Page 15)

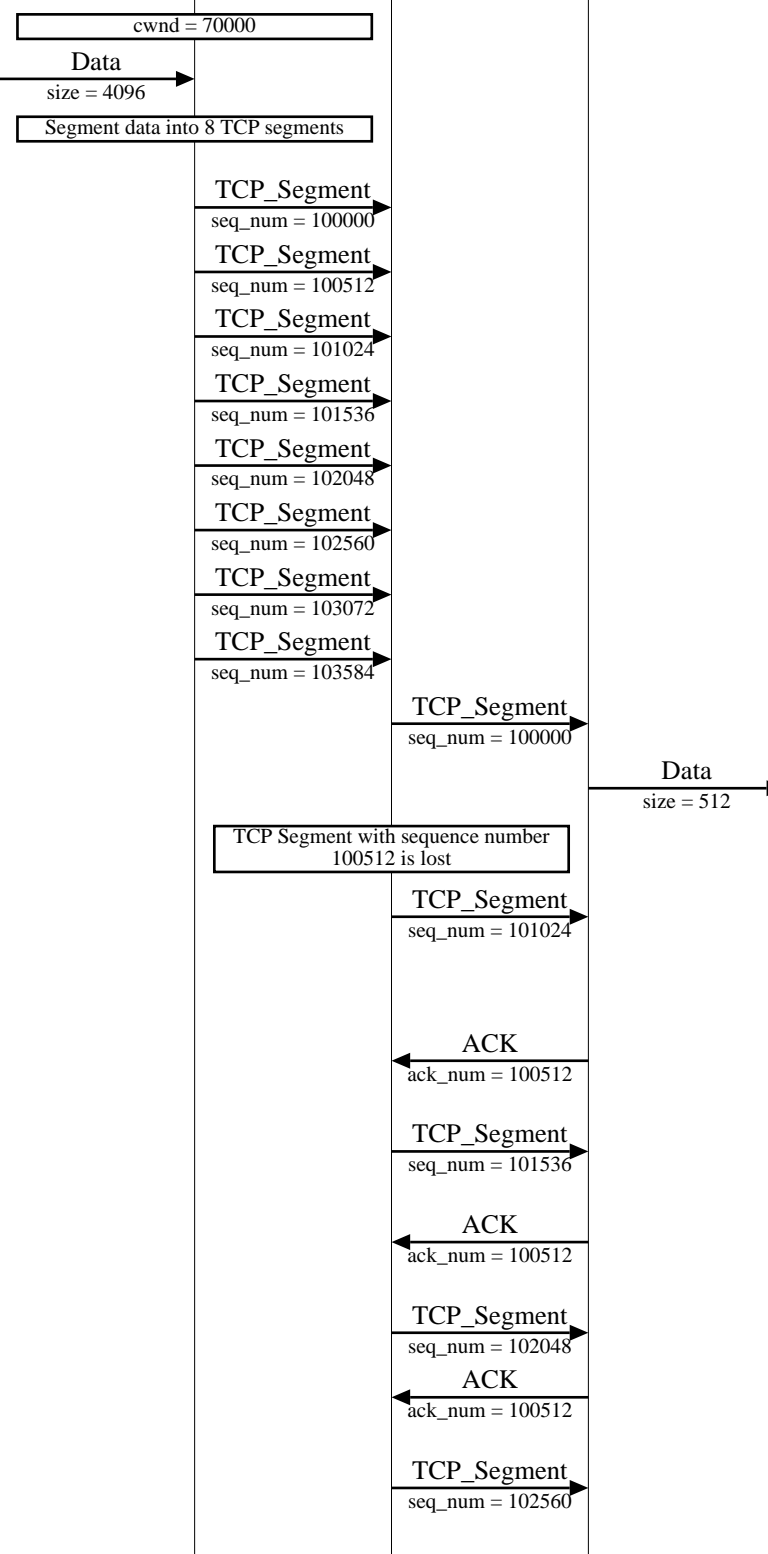


At this point, the client assumes that the TCP connection has been established
Server receives the TCP ACK segment

Now the server too moves to the Established state

LEG: Fast Retransmit and Recovery

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



Congestion window has reached 70000 bytes
Client App transmits 4Kbytes of data

TCP segments data to 8 TCP segments (each segment is 512 bytes)

- TCP segment (start sequence number = 100000) is transmitted
- TCP segment (start sequence number = 100512) is transmitted
- 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 passes 512 bytes of data to the higher layer

TCP segment (start sequence number = 100512) is lost due to congestion

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.

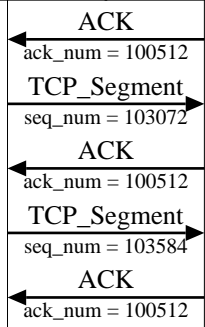
TCP sends an acknowledgement to the Sender with the next expected sequence number set to 100512.

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

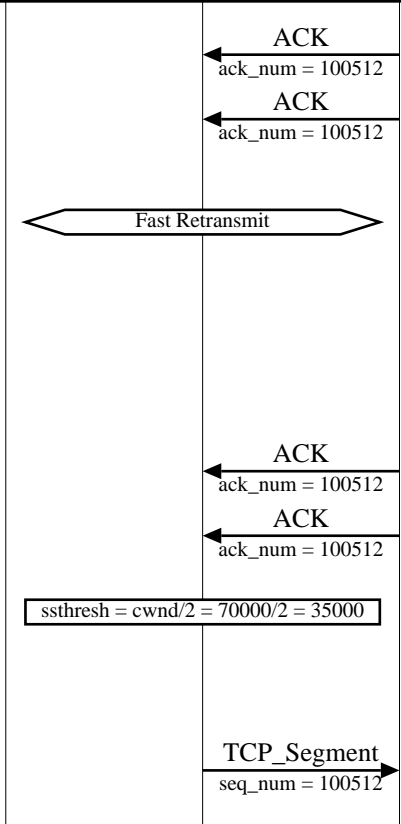
TCP sends another acknowledgement with the next expected sequence number still set to 100512. This is a duplicate acknowledgement

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

TCP - Transmission Control Protocol (TCP Fast Transmit and Recovery)					
Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 16)



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.

At this point TCP moves to the fast retransmit state. TCP will look for duplicate acks to decide if a segment needs to be retransmitted

Note: TCP segments sent by the sender can be delivered out of sequence to the receiver. This can also result in duplicate acks. Thus TCP waits for 3 duplicate acks before concluding that a segment has been missed.

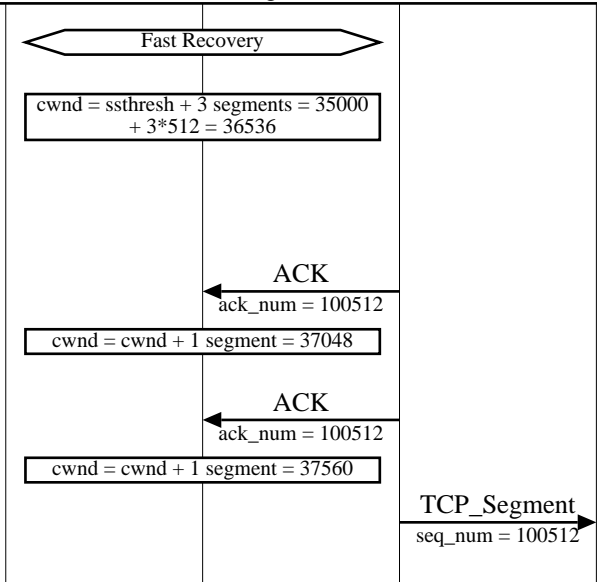
Second duplicate ack is received

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

TCP uses the current congestion window to mark the point of congestion. It saves the slow start threshold as half of the current congestion window size. If current cwnd is less than 4 segments, cwnd is set to 2 segments

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.



In "Fast Recovery" state, TCP's main objective is to maintain the current data stream data flow.

Since TCP started taking action on the third duplicate ack, it sets the congestion window to ssthresh + 3 segment. This halves the TCP window size and compensates for the TCP segments that have already been buffered by the receiver.

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

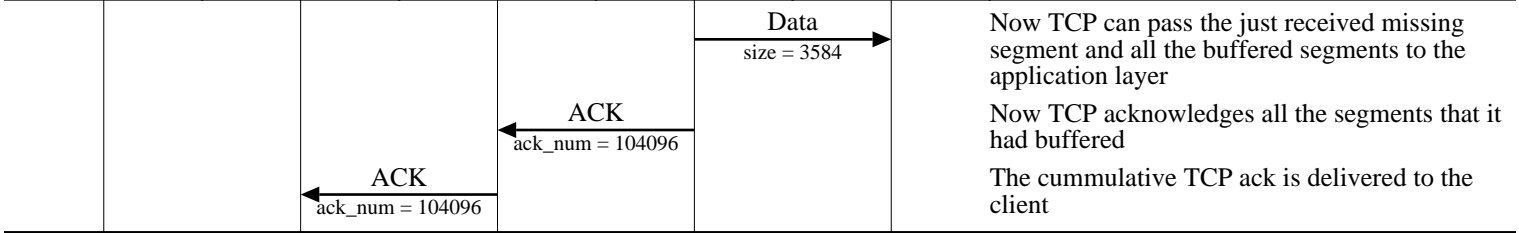
TCP again inflates the congestion window to compensate for the delivered segment

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

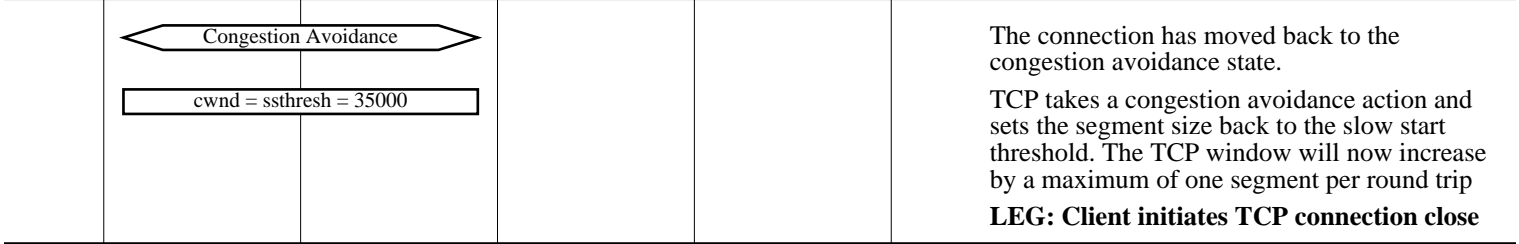
Finally, the retransmitted segment is delivered to the server

TCP - Transmission Control Protocol (TCP Fast Transmit and Recovery)

Client Node		Internet	Server Node		EventHelix.com/EventStudio 2.0
Client		Net	Server		
Client App	Client Socket	Network	Server Socket	Server App	04-Jul-03 14:31 (Page 17)



Congestion Avoidance



Client initiates TCP connection close

