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 socket initialization**

- 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
- Server awaits client socket connections.

**Client socket initialization**

- 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 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

Now the server too moves to the Established state

TCP congestion window grows from 512 bytes (1 segment) to 64947

During slow start the congestion window was being incremented by 1 segment for every TCP Ack from the other end.

Client Application sends data for transmission over the TCP Socket

Data is split into TCP Segments. The segments are sent over the Internet
TCP Segment
seq_num = 100512,
len = 512

TCP Segment
seq_num = 100000,
len = 512

TCP Segment
seq_num = 100512,
len = 512

TCP Segment
seq_num = 101024,
len = 512

TCP Segment
seq_num = 101536,
len = 512

TCP Segment
seq_num = 101024,
len = 512

TCP Segment
seq_num = 101536,
len = 512

Data
Data is forwarded to the server side application

ACK
ack_num = 101024,
window = 80000

ACK
ack_num = 101024,
window = 80000

Data
Data is forwarded to the server application

Client acknowledges the last block and also signals an increase in receiver window to 80000

Since TCP is in slow start, every ack leads to the window growing by one segment.

At this point cwnd (65459) > ssthresh (65535) thus TCP changes state to congestion avoidance. Now TCP window growth will be much more conservative. If no segment or ack losses are detected, the congestion window will grow no more than one segment per roundtrip. (Compare this with geometric growth of 1 segment per TCP ack in slow start)

More data is received from the client application

Client data is split into TCP segments
Now TCP is in congestion avoidance mode, so the TCP window advances very slowly. Here the 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 does not play a 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.

First part of the data is delivered.

Second Part of the data is delivered.
TCP Segment size = 512
TCP Segment size = 512
Data size = 1024
Third Part of the data is delivered

ACK

Ack for the first two segments is received
TCP is in slow start so the congestion window is increased by one segment

ACK

Now cwnd (=33242") > ssthresh (=32731), thus the TCP session moves into congestion avoidance

ACK

Ack for the next two segments is received
Now the TCP window is growing very slowly by approximately 8 bytes per ack

ACK

Ack for the last two segments is received
Congestion window continues to advance at a slow rate

Client closes TCP connection

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 to client TCP connection release

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.

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