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.

**LEG: About TCP Slow Start**

**Client initiated three-way handshake to establish a TCP connection**

**Client initiated three-way handshake to establish a TCP connection**

**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 the 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**
TCP - Transmission Control Protocol (TCP Slow Start)

TCP慢启动（TCP Slow Start）

TCP连接开始时处于“慢启动”状态。这意味着TCP根据从对方端接收的应答的速率来调整其传输速率。

TCP慢启动通过两个变量实现，即cwnd（拥塞窗口）和ssthresh（慢启动阈值）。cwnd是发送方自定义的传输窗口限制。

当cwnd增加到ssthresh时，TCP会考虑该方向的连接从慢启动阶段退出。

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.

TCP慢启动是通过两个变量实现的，即cwnd（拥塞窗口）和ssthresh（慢启动阈值）。cwnd是发送方定义的传输窗口限制。cwnd会随着TCP对网络处理流量的信心增加而增加。ssthresh是确定TCP退出慢启动阶段的阈值。如果cwnd增加超过ssthresh，TCP在该方向的连接将被认为是退出慢启动阶段。

Data size = 5120

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.
### Roundtrip #2 of data transmission

Client receives the acknowledgement for the first TCP data segment:

\[
\text{rwnd} = 1024 \text{ (2 segments)}
\]

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

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

Receiver generates a TCP ACK on receiving the two segments.

Receipt for ack again moves the congestion window.

### Roundtrip #3 of data transmission

Now three segments can be sent without waiting for an ack.

Network delivers the three segments to the destination server.

TCP acknowledges receipt of two segments.
TCP times for another segment and acknowledges the only pending segment.

The TCP acknowledgements 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.

The 5 segments are received by the destination server.

TCP Ack is sent after first two segments.

Ack for next two segments

Ack for last segment
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.

Roundtrip #5 of data transmission

This time 8 TCP segments are sent

Ack for first two segments

Ack for next two segments

Ack for next two segments

Ack for next two segments

Now four acks will be received, thus moving cwnd even more quickly
TCP - Transmission Control Protocol (TCP Slow Start)

Client Node | Internet | Server Node
--- | --- | ---
Client | Net | Server
Client App | Client Socket | Network | Server Socket | Server App

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 network's capacity. From this point on, the congestion window will not move geometrically. cwnd will move linearly once the connection is out of slow start.

Congestion Avoidance

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.