TCP

Tom Kelliher, CS 325

Apr. 2, 2008

1 Administrivia

Announcements

Assignment

Read 3.6–7.

From Last Time

Web server and mail user agent project discussions.

Outline

1. TCP connection and segment structure.
2. Round trip delay estimation.
3. Reliable data transfer.
4. Flow control.
5. Connection management.

Coming Up

Congestion control.

2 The TCP Connection and Segment Structure

Recall that:

1. TCP is connection-oriented — three-step "handshake."

2. TCP state only resides in the source and destination hosts — not within intermediate hosts.

3. TCP is full-duplex and point-to-point.

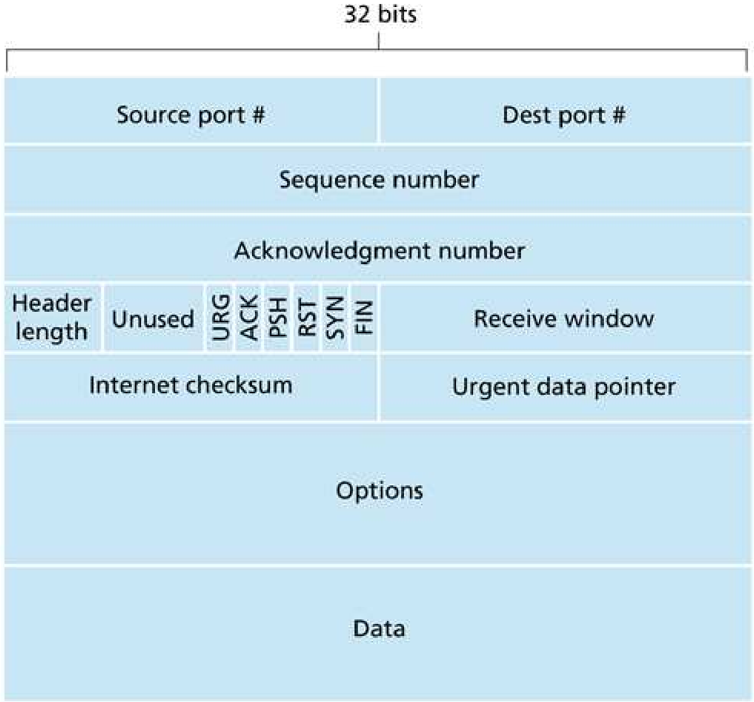
4. Maximum segment size (MSS) is limited by maximum transmission unit (MTU), which is the largest link-level frame that can be sent.

   MSS is data only. Path MTU discovery.

5. A segment consists of TCP header information and the data.

6. TCP connection state: send/receive buffers, variables, socket.

Segment structure:



1. Receive window: Used for flow control. Number of bytes a receiver is willing to accept.

2. URG: Upper layer sending protocol has marked this data as "urgent."

3. ACK: Indicates that the acknowledgement field is valid.

4. PSH: Receiver should push data to upper layer protocol immediately.

5. RST: Reset the connection — segment sent to a non-existent socket.

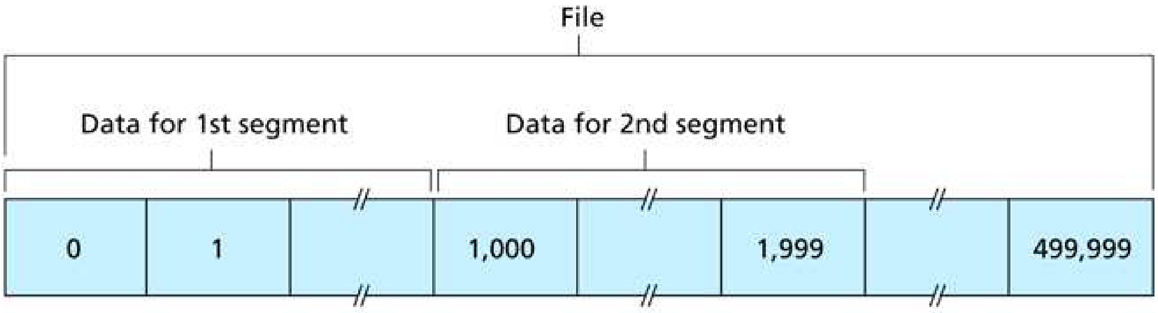
6. SYN: Used during initial handshake.

7. FIN: Used during connection tear-down.

8. Urgent Data Pointer: Pointer to last byte of urgent data.

Segment and acknowledgement numbers:

1. Individual bytes have sequence numbers:

   File size of 500 KB with an MSS of 1,000 B.

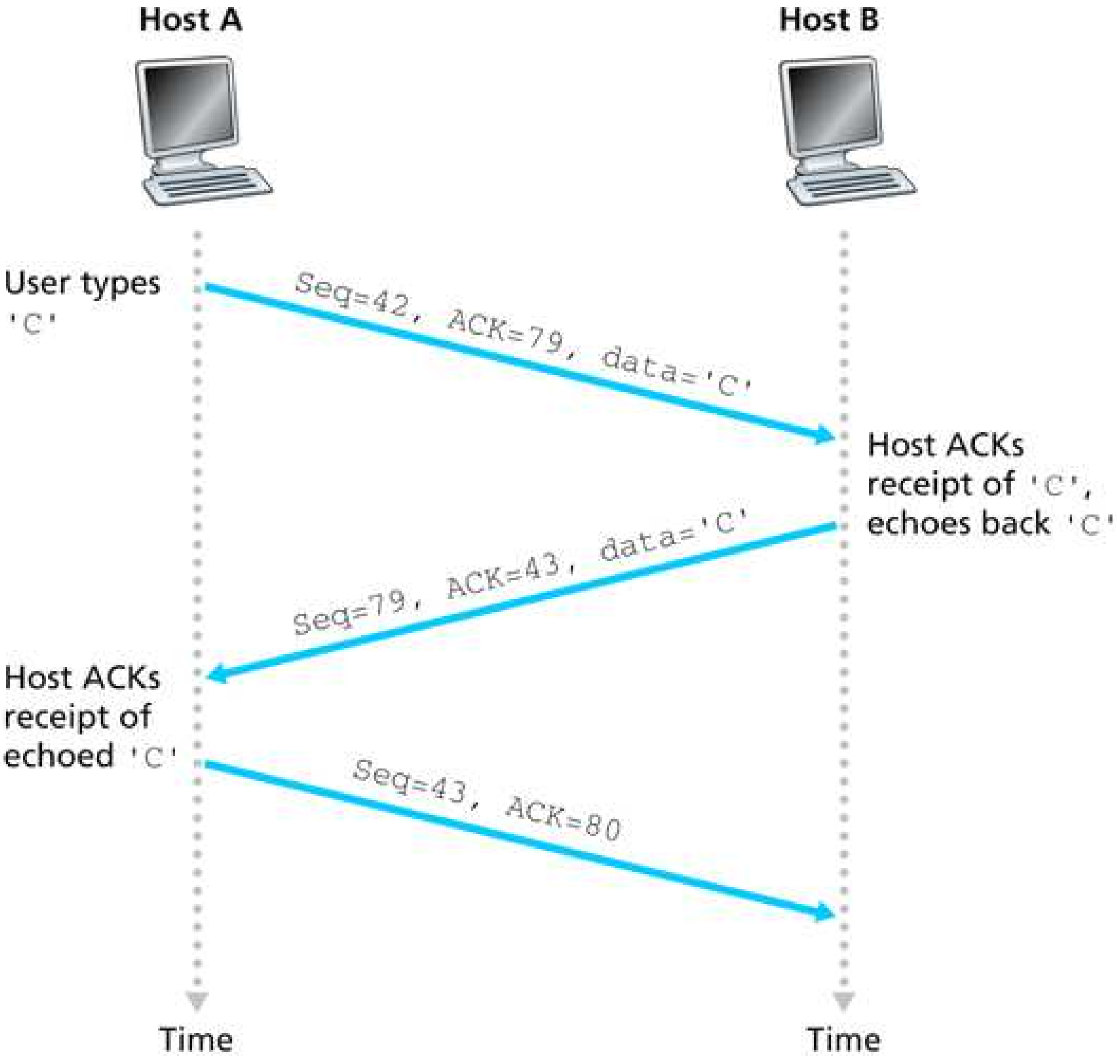
   Segment number for a segment is segment number of first byte.

   Acknowledgement number is **next** expected segment number.

2. Acknowledgements are cumulative.

   A receiver will accept segments out of order, but will not acknowledge them if earlier segments have not been received.

3. Telnet example (next sequence number of client is 42; for server, it's 79):



Remote host handles echoing. Response time crucial for interactive applications.

3 Round Trip Delay Estimation

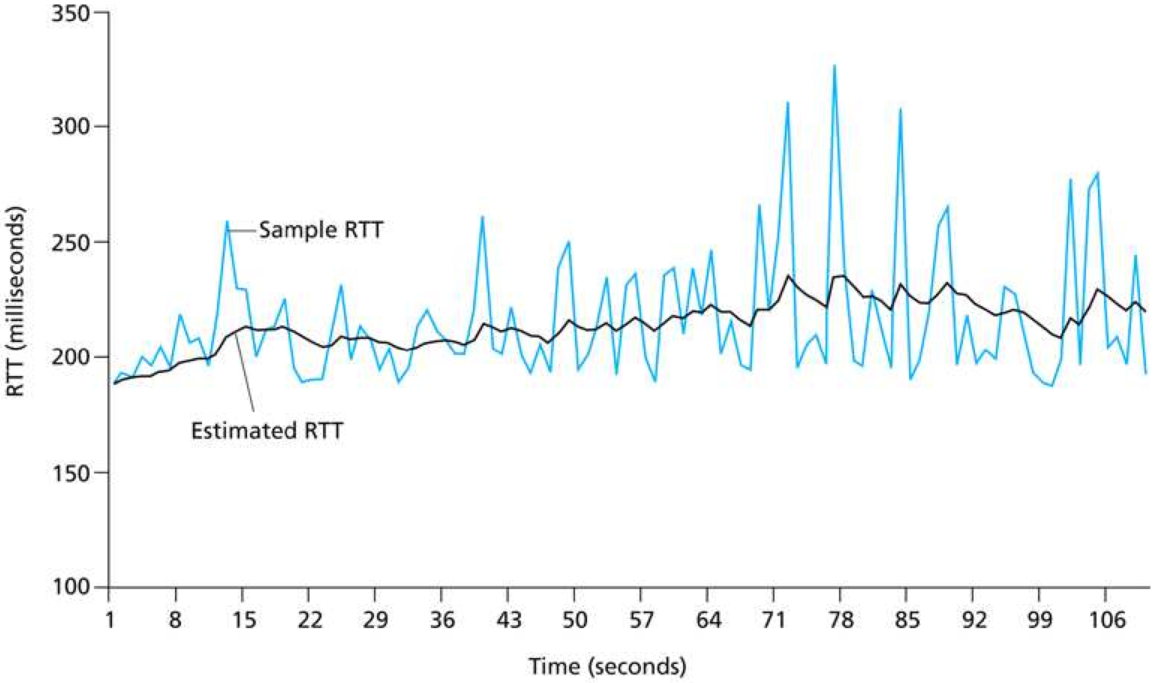
1. TCP uses timeout/retransmission. How is the timeout interval determined?

2. `SampleRTT`: Measurement of a sample RTT. Typically, one done at a time.

3. `EstimatedRTT` is an exponential weighted average:

$$\texttt{EstimatedRTT} = 0.875 \times \texttt{EstimatedRTT} + 0.125 \times \texttt{SampleRTT}$$

4. Relationship between `SampleRTT` and `EstimatedRTT`:



5. Also need to account for variance in RTTs. `DevRTT` estimates the variance:

$$\texttt{DevRTT} = 0.75 \times \texttt{DevRTT} + 0.25 \times |\texttt{SampleRTT} - \texttt{EstimatedRTT}|$$

6. Finally, the `TimeoutInterval` needs to provide some cushion to prevent unnecessary retransmissions:

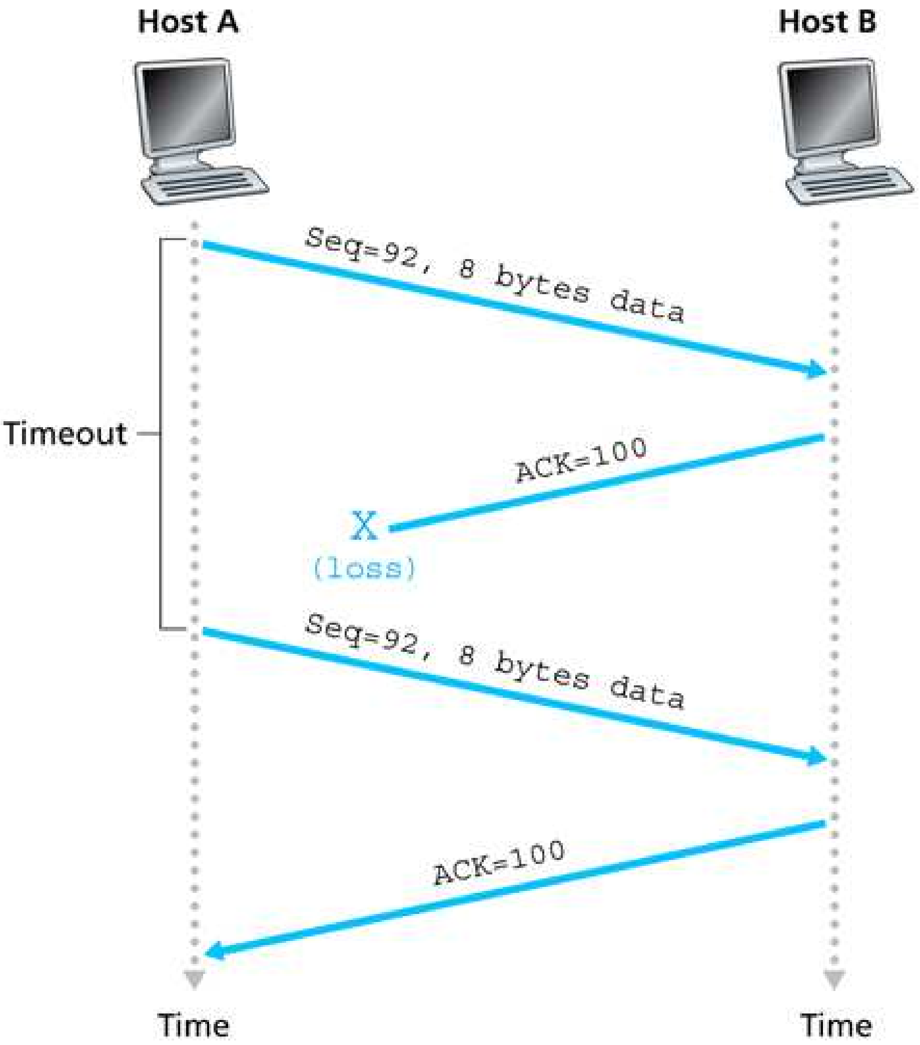
$$\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4 \times \texttt{DevRTT}$$

4 Reliable Data Transfer

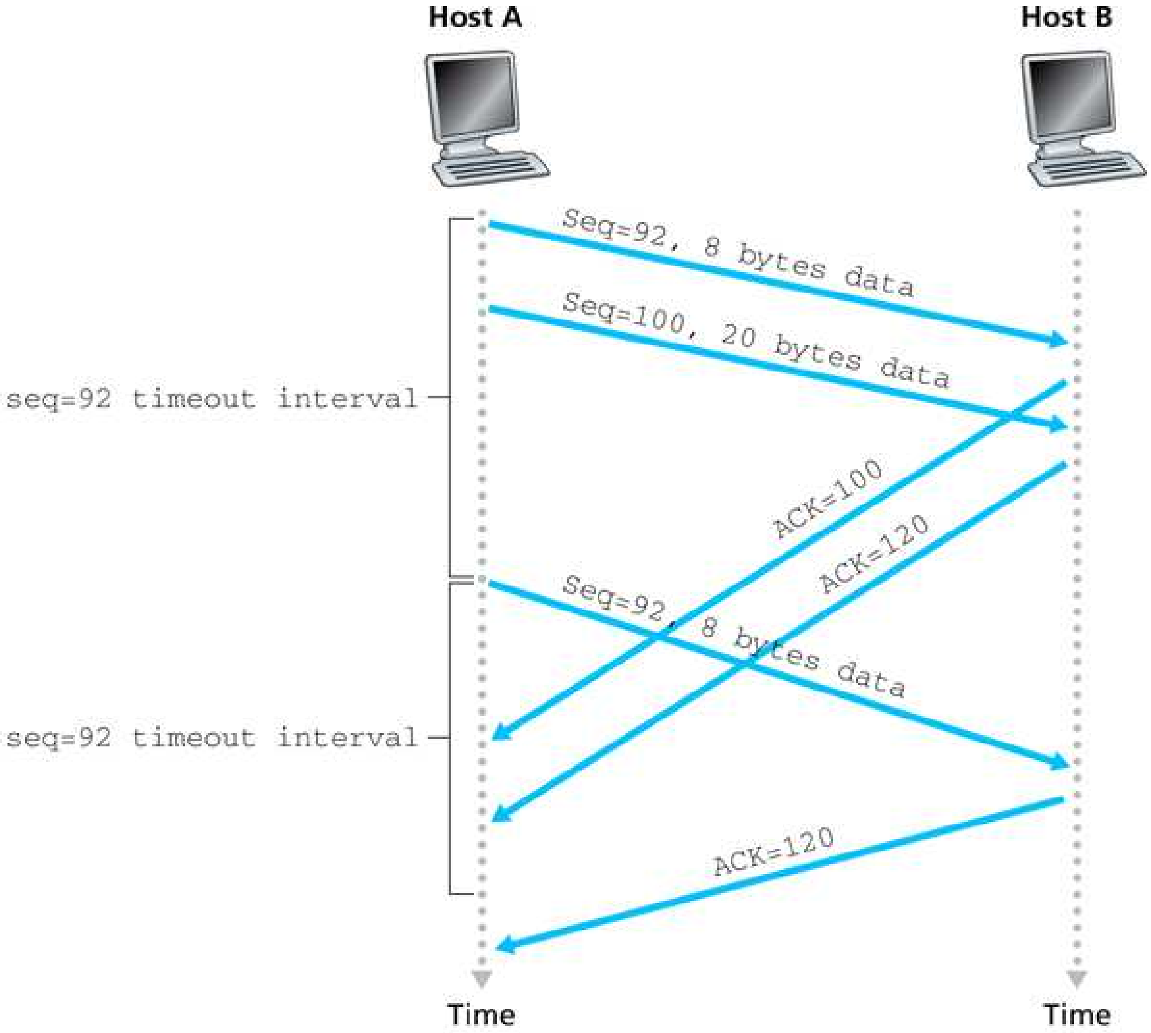
Simplified TCP sender:

```
NextSeqNum = InitialSequenceNumber;   // Must be pseudo-randomly chosen.
SendBase = InitialSequenceNumber;

while (1)
{
   switch(event)
   {
      case: DataReceivedFromApplicationAbove
         create TCP segment with sequence number NextSeqNum;
         if (TimerNotRunning)
            start timer;   // Use TimeoutInterval value.
         pass segment to IP
         NextSeqNum += length(Data);
         break;

      case: TimerTimeout
         retransmit not-yet-acknowledged segment
            with smallest segment number;
         start timer;   // Double timeout interval.
         break;

      case: ACKReceivedWithACKFieldValueOfY
         if (y > SendBase)
         {
            SendBase = y;
            if (UnAcknowledgedSegmentsExist)
               start timer;   // Use TimeoutInterval value.
         }
         break;
   }
}
```

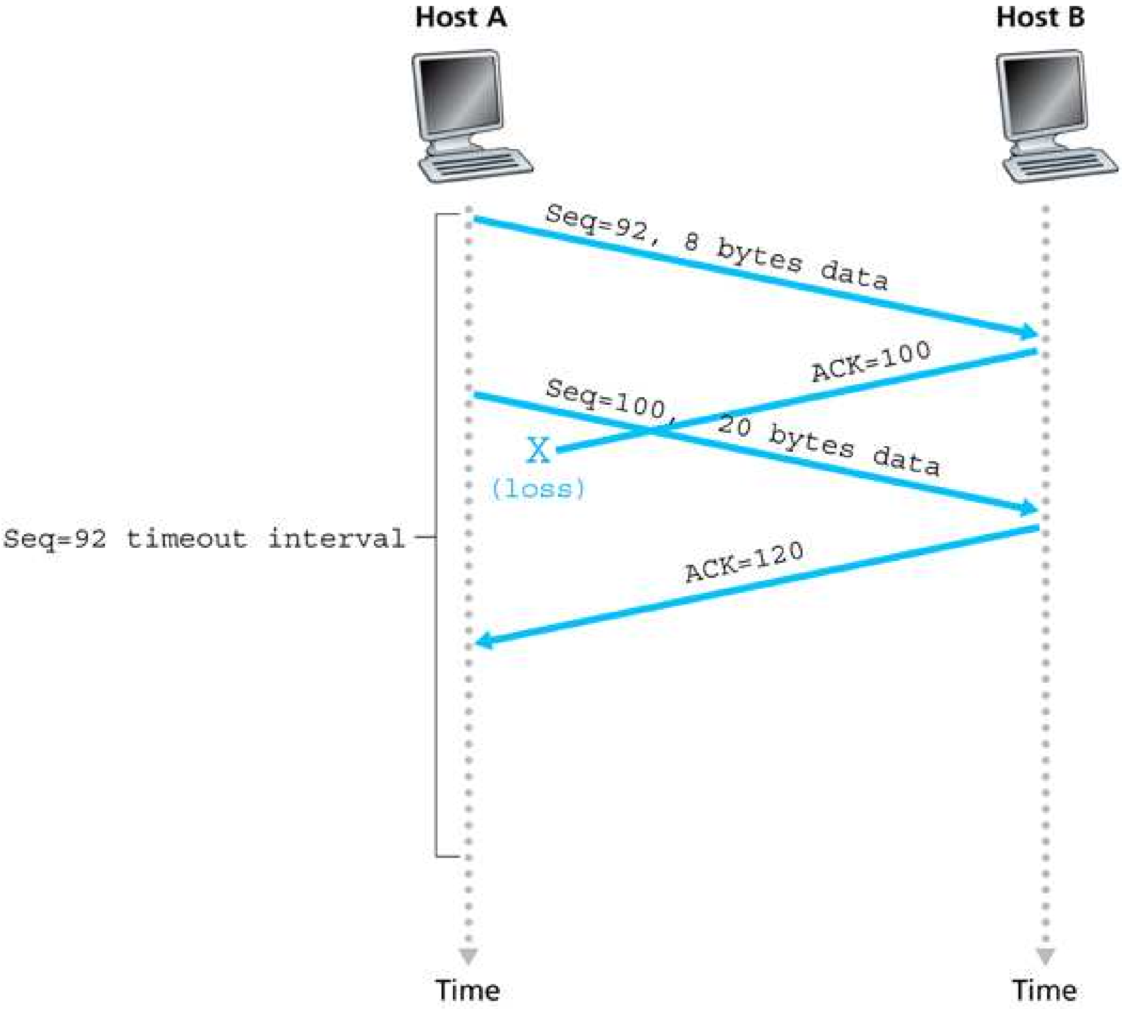
Example — retransmit due to lost ACK:



Example — only oldest segment gets retransmitted.



Example — cumulative acknowledgement handles lost ACK:



Fast retransmit:

1. Sender can detect lost segments before timer expiration by looking for duplicate ACKs of an “older” segment.

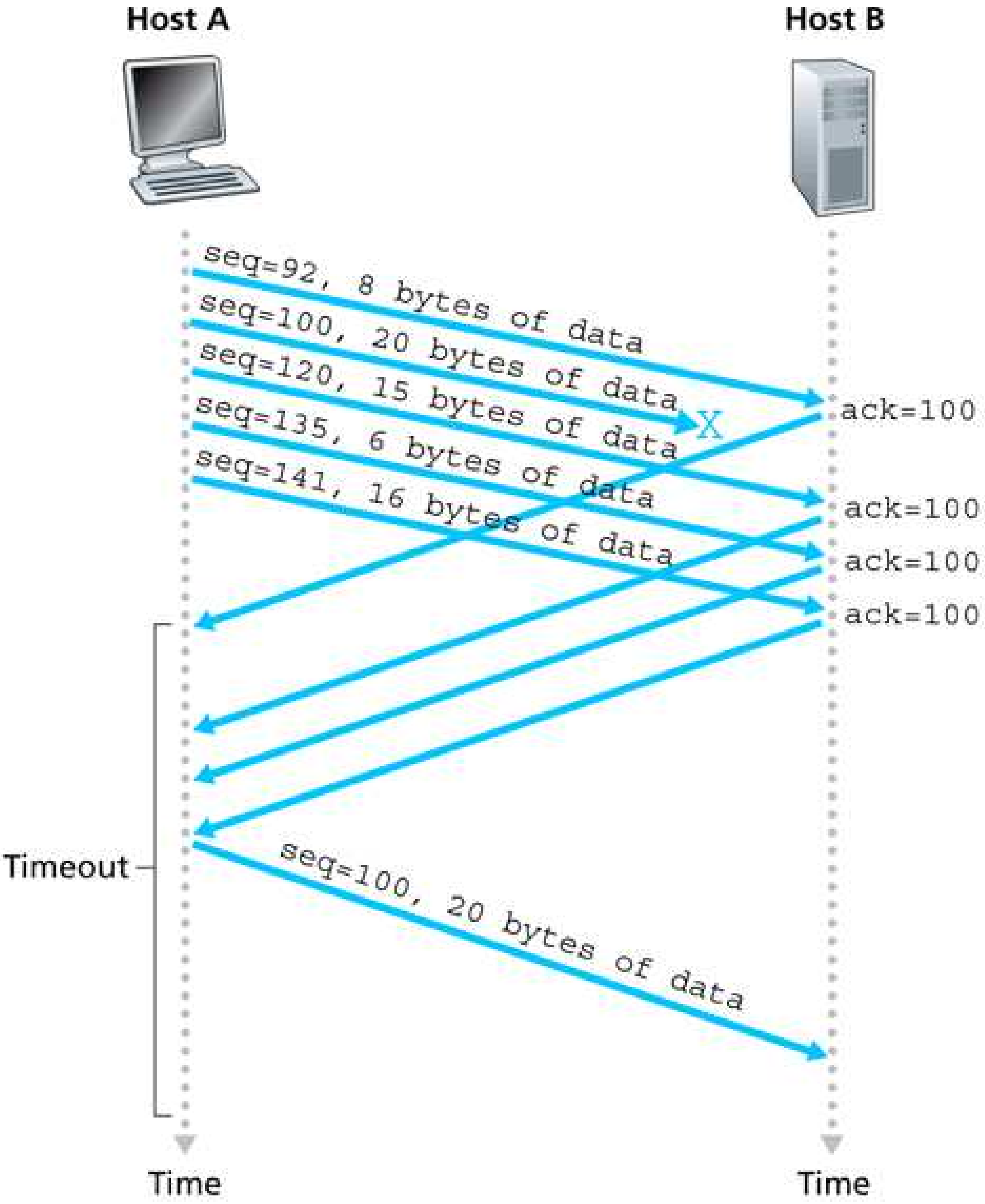
   Only takes effect if three duplicate ACKs are received for a segment:

   ```
   case: ACKReceivedWithACKFieldValueOfY
      if (y > SendBase)
      {
         SendBase = y;
         if (UnAcknowledgedSegmentsExist)
            start timer;   // Use TimeoutInterval value.
      }
      else   // Duplicate ACK.
      {
         IncrementDuplicateACKCount(y);
         if (DuplicateACKCount(y) == 3)
            resend segment y;
      }
      break;
   ```

2. Receiver behavior used to provide “hints” to sender:

Event	**TCP Receiver Action**
Arrival of in-order segment with expected sequence number. All segments up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 ms for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment wait for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. **Gap detected.**	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Fast retransmit example:



5 Flow Control

Don't confuse with congestion control!

1. Allows receiver to throttle sender to match consumption rate of process bound to socket.

2. There is a RcvWindow field in the TCP header.

3. Each receiver computes the size of its receive window and sends it with TCP segments:

   $$\texttt{RcvWindow} = \texttt{RcvBuffer} - (\texttt{LastByteRcvd} - \texttt{LastByteRead})$$

   So, RcvWindow is the amount of space available in the receive buffer.

4. Sender decides how much data it can send by:

   $$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{RcvWindow}$$

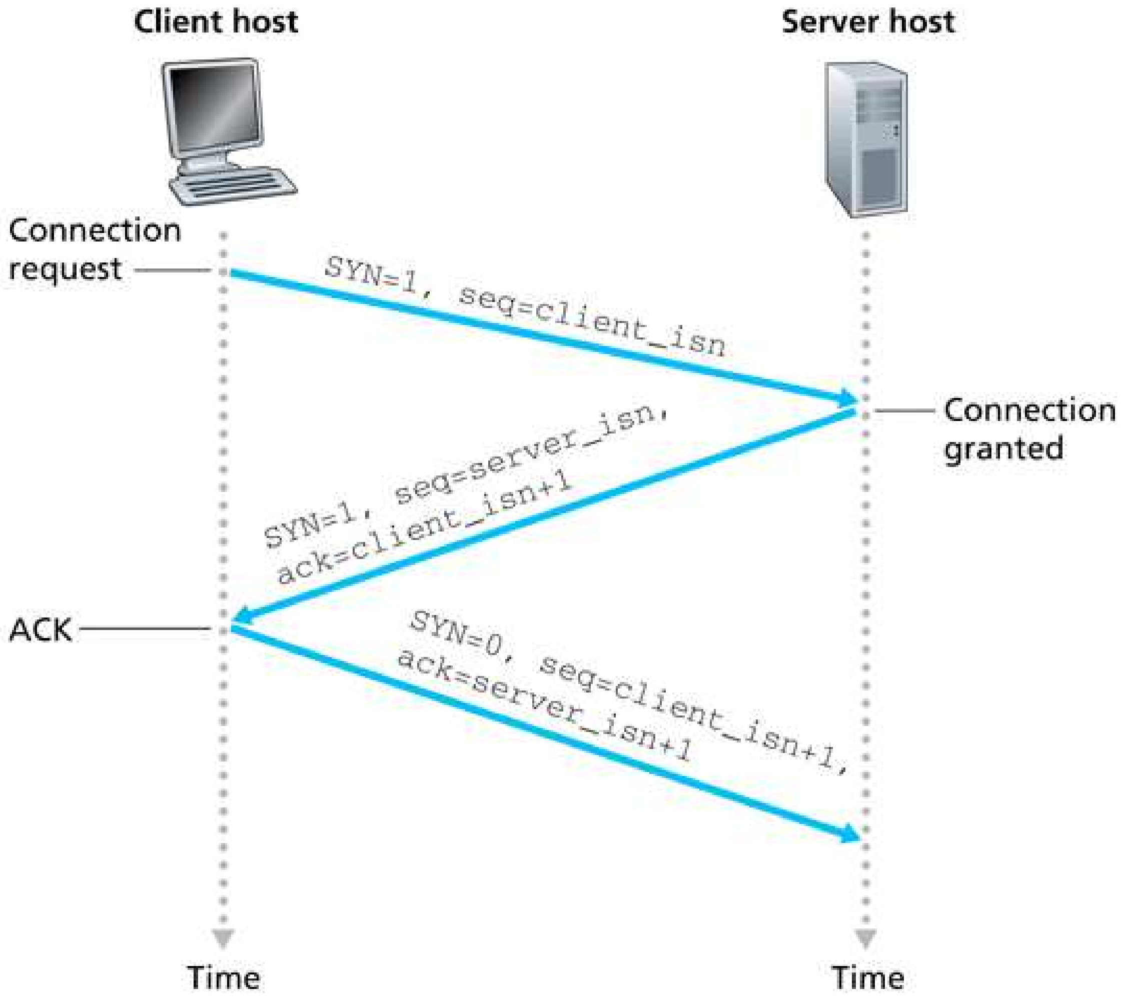
   LHS is an idea of how much data is "in the pipe."

5. Dilemma: What does sender do when receive window is 0?

   Solution: Send segments with one byte of data, so as to receive updates as to current receive window size.

6 Connection Management

The three-way handshake:



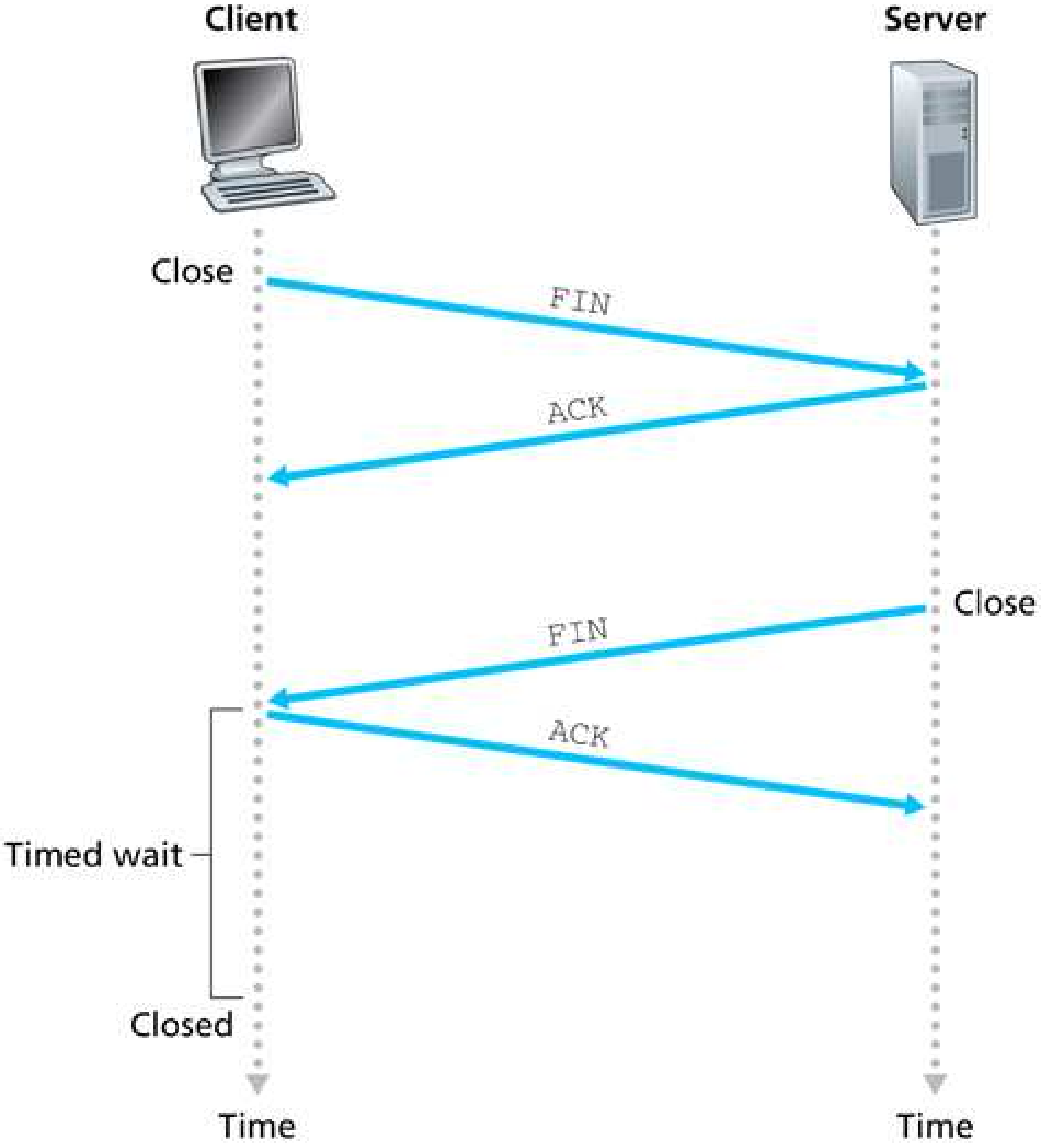
1. Client chooses **random** ISN. Sends SYN segment.

2. Server allocates state for connection. Selects ISN. Sends SYNACK segment.

   Server vulnerable to SYN flooding at this point.

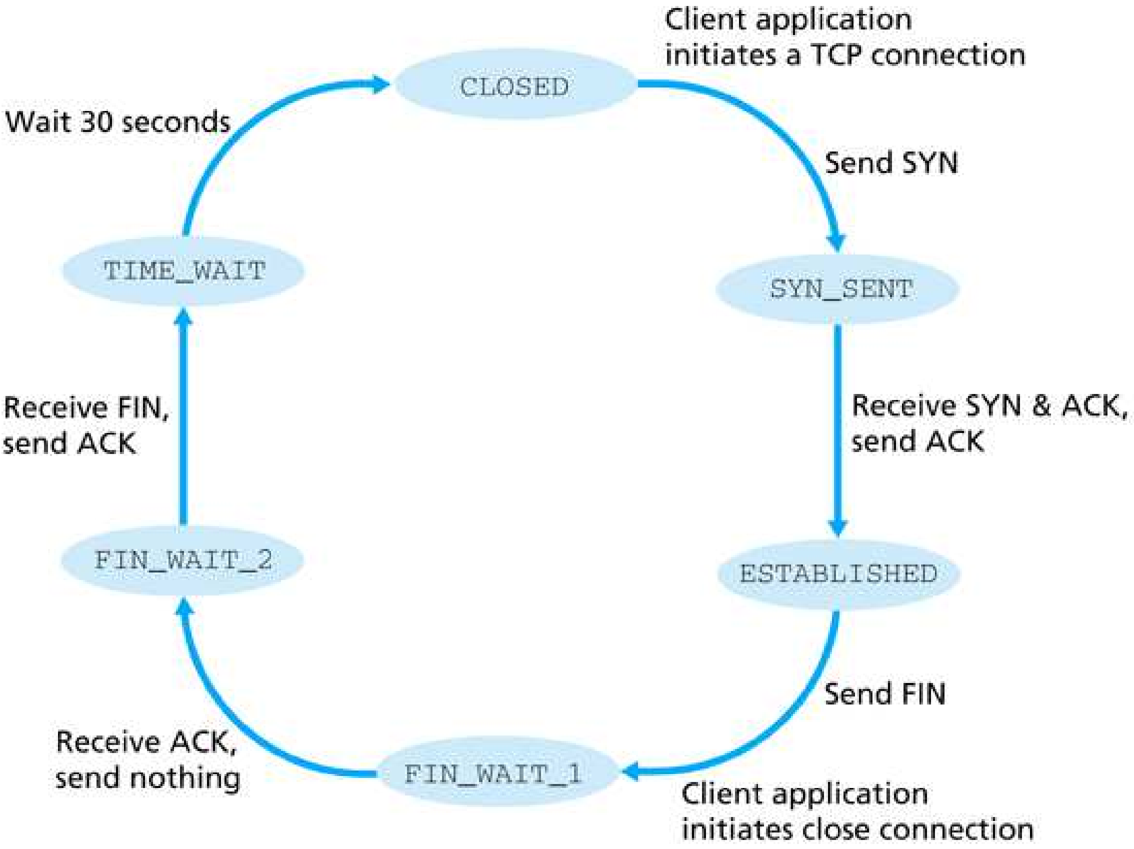
3. Client allocates state and ACKs server's SYN segment.

Client closes the connection:



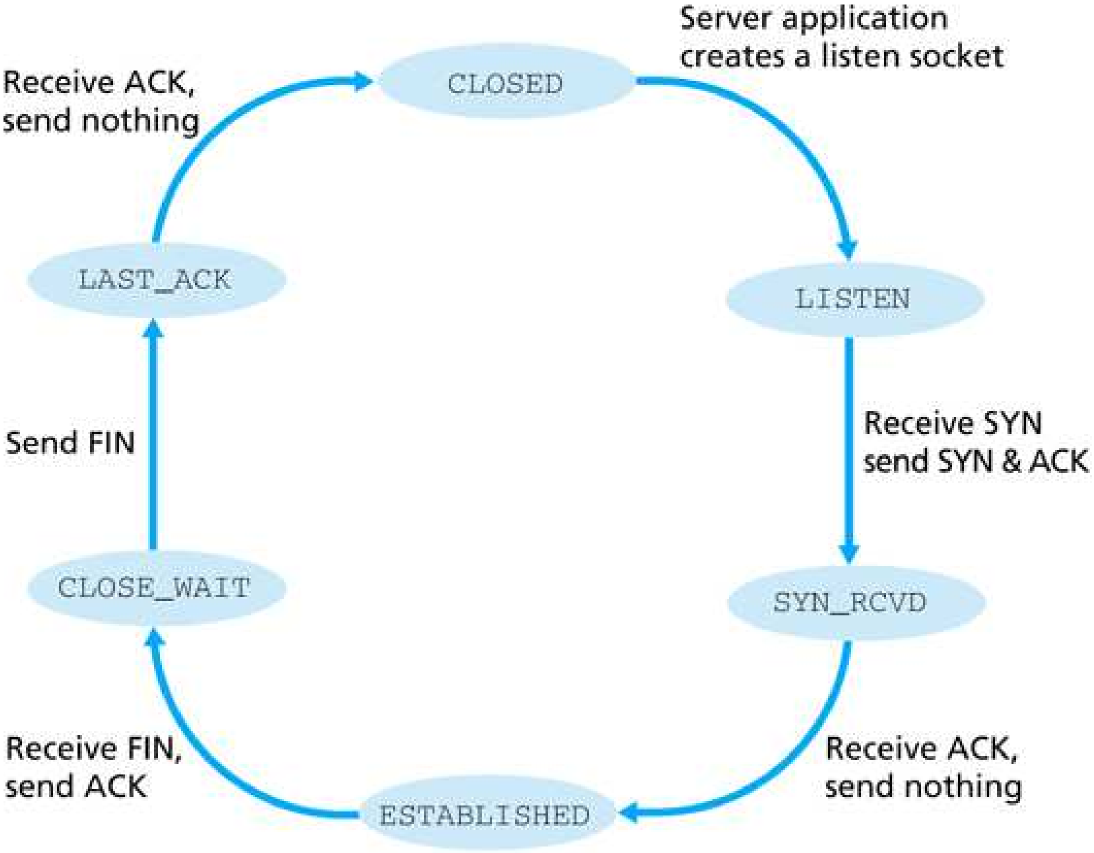
Timed Wait: Client keeps connection "around" in case it needs to resend ACK to server's FIN.

Typical sequence of client TCP states:



This assumes the client begins the connection close sequence.

Typical sequence of server TCP states:


CLOSED
Server application creates a listen socket
LISTEN
Receive SYN send SYN & ACK
SYN_RCVD
Receive ACK, send nothing
ESTABLISHED
Receive FIN, send ACK
CLOSE_WAIT
Send FIN
LAST_ACK
Receive ACK, send nothing

Again, this assumes the client begins the connection close sequence.