Transport Layer

Instructor: C. Pu (Ph.D., Assistant Professor)

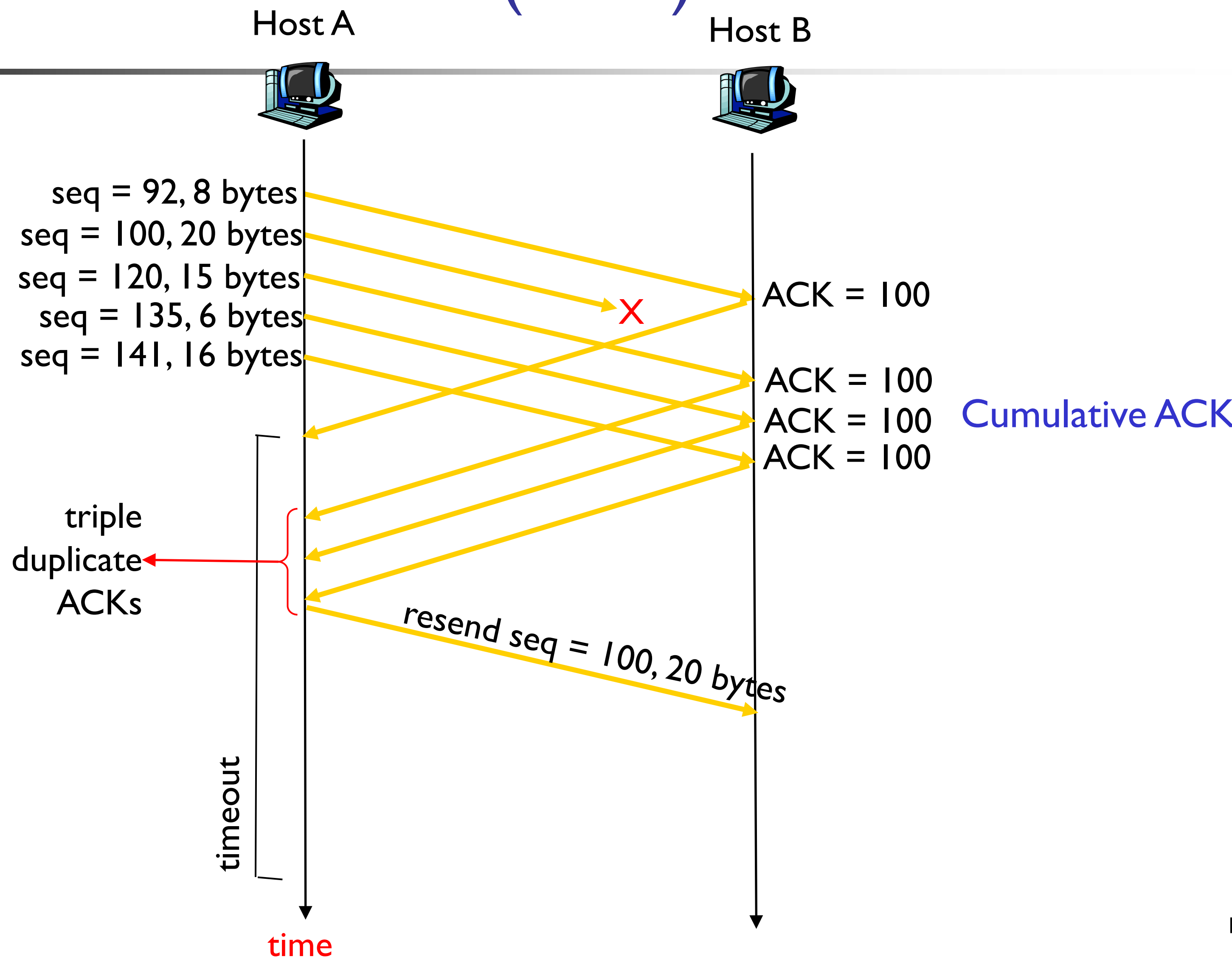
Lecture 12

puc@marshall.edu

Fast Retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
  - increase end-to-end delay
- detect lost segments via duplicate ACKs
  - duplicate ACKs: ACK that reAcks a segment for which the sender has already received an earlier Ack
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs
- if sender receives **3 ACKs** for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend unacked segment with smallest seq # before timeout
  - likely that unacked segment lost, so don't wait for timeout

Fast Retransmit (cont.)



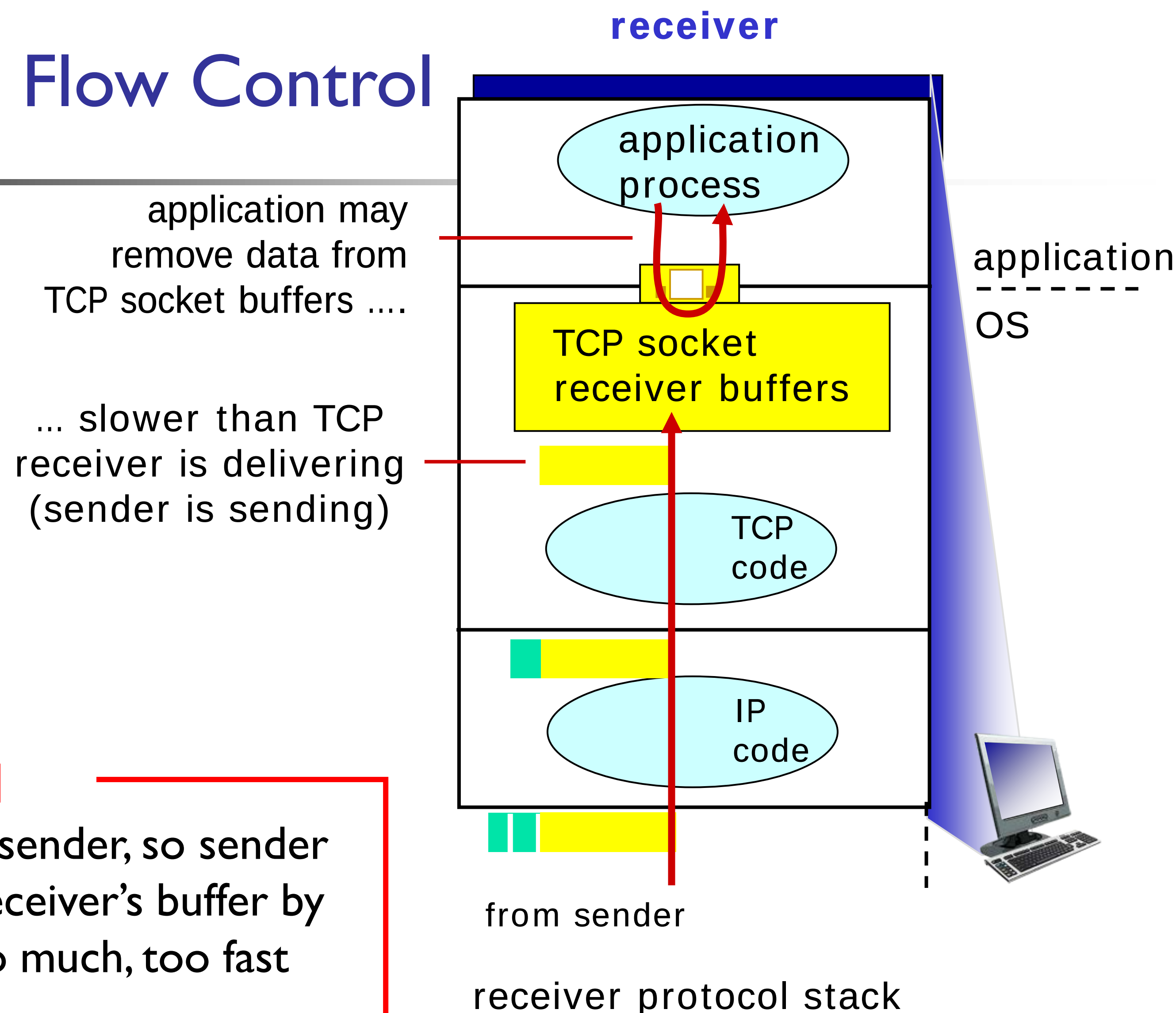
Fast Retransmit (cont.)

```
event: ACK received, with ACK field value of y
          if (y > SendBase) {
             SendBase = y;
             if (there are currently not-yet-acknowledged segments)
                start timer;
          }
          else {
              increment count of dup ACKs received for y;
              if (count of dup ACKs received for y == 3) {
                  resend segment with sequence number y;
          }
```

a duplicate ACK for already ACKed segment

fast retransmit

TCP Flow Control



flow control

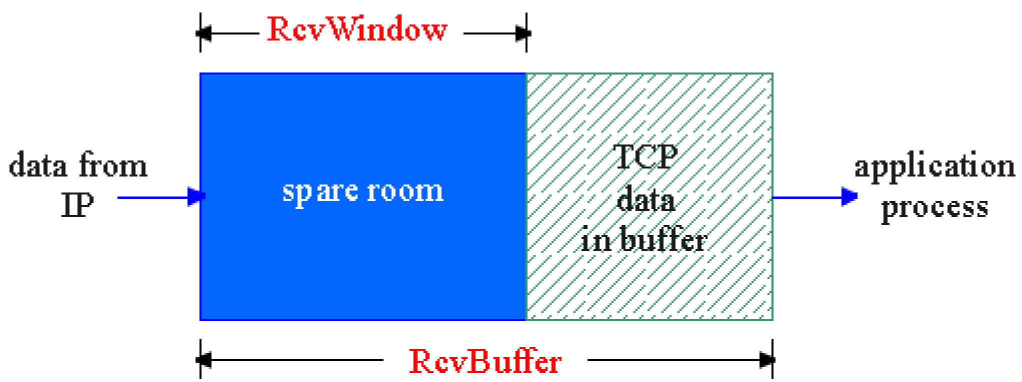
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

TCP Flow Control (cont.)

flow control

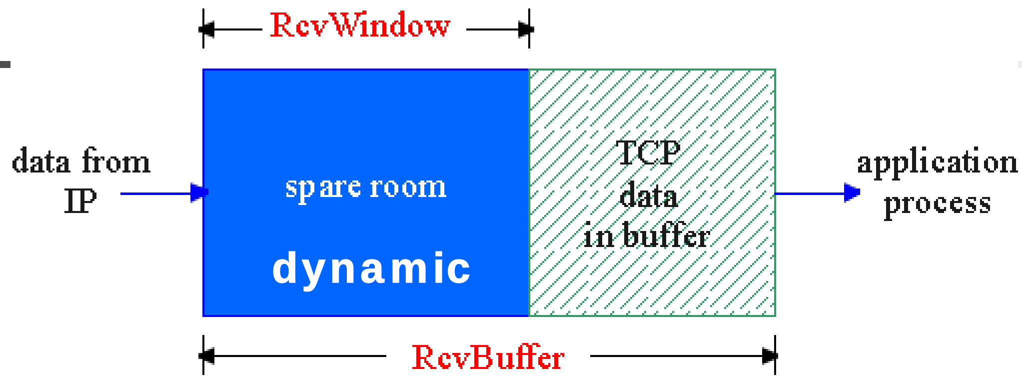
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

- receiver side of TCP connection has a receive buffer:



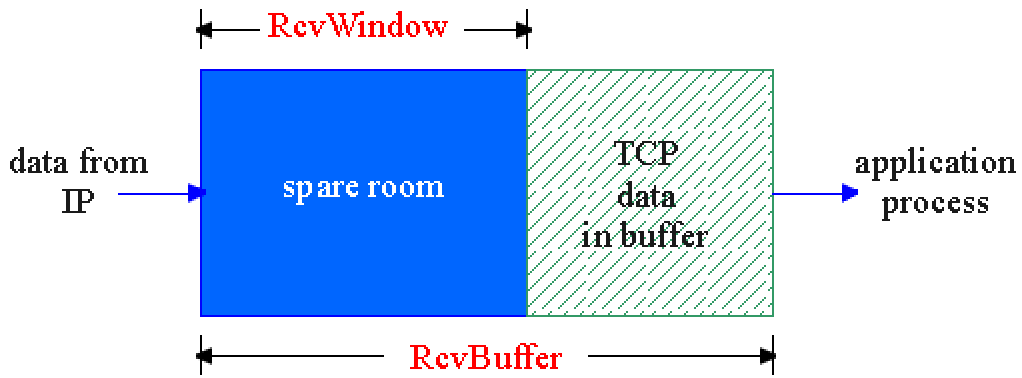
- app. process may be slow at reading from buffer
- speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow Control (cont.)



- spare room in buffer
  - `LastByteRcvd – LastByteRead <= RcvBuffer`
  - `RcvWindow (rwnd) = RcvBuffer -[LastByteRcvd - LastByteRead]`
- receiver advertises spare room by including value of **rwnd** in TCP header receiver-to-sender segments
  - RcvBuffer size set via socket options (typical default is 4096 bytes)
- sender limits amount of unACKed ("in-flight") data to receiver's **rwnd** value
  - guarantees receive buffer will not overflow
  - `LastByteSent – LastByteAcked <= rwnd`

TCP Flow Control (cont.)



- what if,
  - Host B's receive buffer is 0 (rwnd = 0)
  - Host B has nothing to send to A
    - Host A is never informed about rwnd
- Host A is blocked and can transmit no more data!! **How to solve?**
  - Host A continues to send segments with one data byte, when B's receive window is zero

TCP Connection Management

- recall:
  - TCP sender and receiver establish "connection" before exchanging data segments
  - agree to establish connection and agree on connection parameters
- initialize TCP variables:
  - seq. #s
  - buffers
  - flow control info (e.g. **rwnd**)

TCP Connection Management

- how a TCP connection is established
  - suppose a process running in one host (client) wants to initiate a connection with another process in another host (server)
  - the client application process first informs the client TCP that it wants to establish a connection to a process in the server
  - the TCP in the client then proceeds to establish a TCP connection with the TCP in the server in the following manner:
    1. the client-side TCP first sends a *special TCP segment* to the server-side TCP
       - this special segment contains no application-layer data
       - the SYN bit, is set to 1 (SYN segment)
       - the client randomly chooses an initial sequence number (client_isn) and puts this number in the sequence number field of the initial TCP SYN segment

TCP Connection Management

- how a TCP connection is established
  2. once the first segment arrives at the server host
     - extracts the *TCP SYN segment* from the datagram
     - allocates the TCP buffers and variables to the connection
     - sends a connection-granted segment to the client TCP
     - no application data
     - SYN bit is set to 1
     - the acknowledgment field of the TCP segment header is set to client_isn + 1
     - the server chooses its own initial sequence number (server_isn) and puts this value in the sequence number field of the TCP segment header
     - the connection-granted segment is called *SYNACK segment*

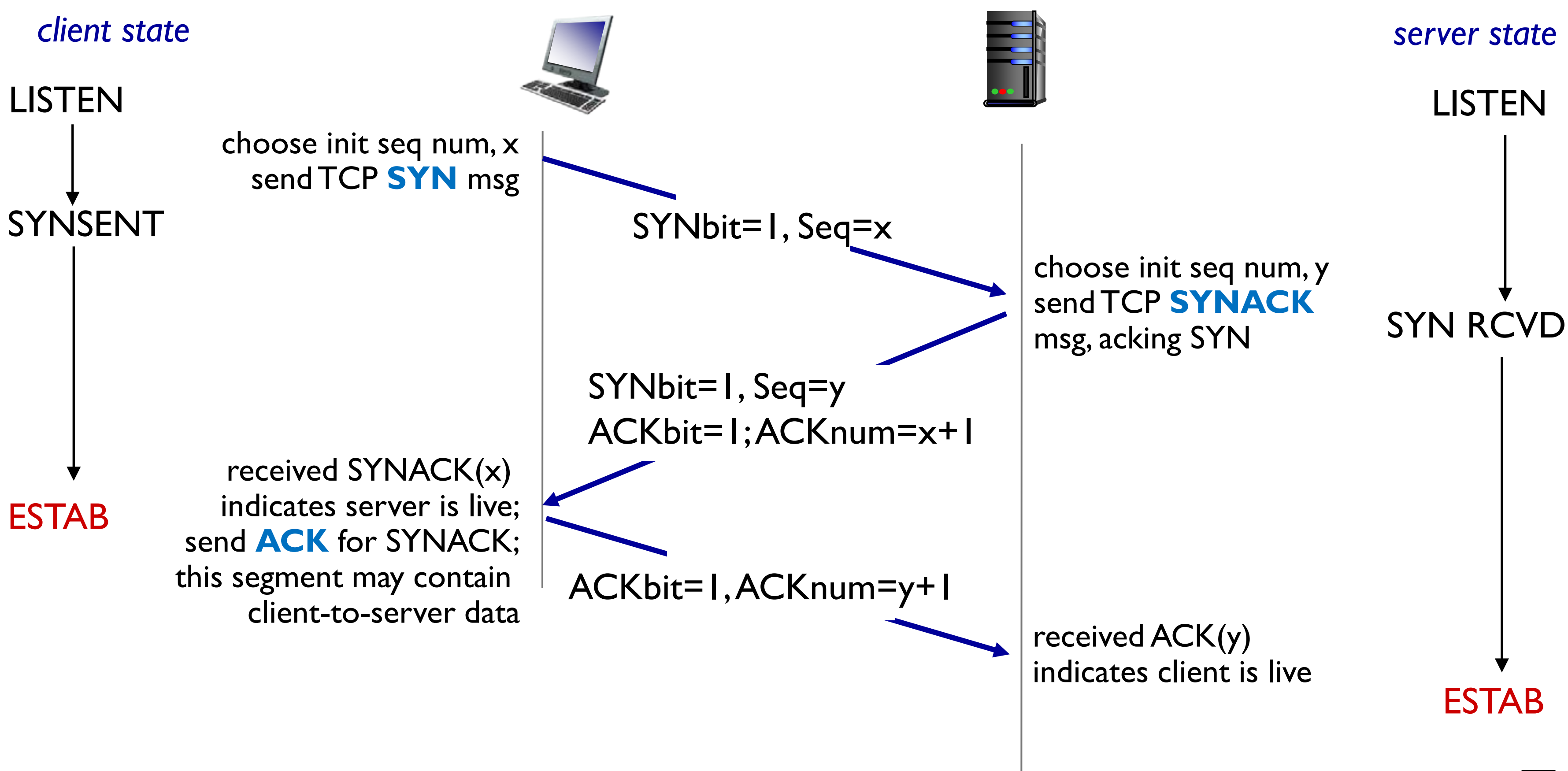
TCP Connection Management

- how a TCP connection is established
  3. once the SYNACK segment arrives at the client host
     - the client also allocates buffers and variables to the connection
     - the client host then sends the server yet another segment
       - this last segment acknowledges the server's connection-granted segment (putting the value server_isn + 1 in the acknowledgment field of the TCP segment header)
     - the SYN bit is set to zero, since the connection is established
     - this third stage of the three-way handshake may carry client-to-server data in the segment payload

TCP Connection Management

- how a TCP connection is established
  - once these three steps have been completed, the client and server hosts can send segments containing data to each other
  - in each of these future segments, the SYN bit will be set to zero
  - note that in order to establish the connection, three packets are sent between the two hosts
    - *three-way handshake*

TCP 3-Way Handshake



TCP Connection Management

- how a TCP connection is torn down
  - either of the two processes participating in a TCP connection can end the connection
  - when a connection ends, the "resources" (that is, the buffers and variables) in the hosts are deallocated
  - suppose the client decides to close the connection
    - the client TCP to send a special TCP segment to the server process
      - this special segment has a flag bit in the segment's header, the FIN bit, set to 1
    - when the server receives this segment, it sends the client an acknowledgment segment in return
    - the server then sends its own shutdown segment, which has the FIN bit set to 1
    - finally, the client acknowledges the server's shutdown segment
    - at this point, all the resources in the two hosts are now deallocated

TCP: Closing a Connection (cont.)