UIC UNIVERSITY OF ILLINOIS AT CHICAGO Department of Computer Science
COLLEGE OF ENGINEERING

# CS450 – Introduction to Networking
## Lecture 16 – TCP (2)

Phu Phung

Feb 18, 2015

# TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

let's initially consider simplified TCP sender:

- ignore duplicate acks
- ignore flow control, congestion control

# TCP sender events:

*data rcvd from app:*

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval: `TimeOutInterval`

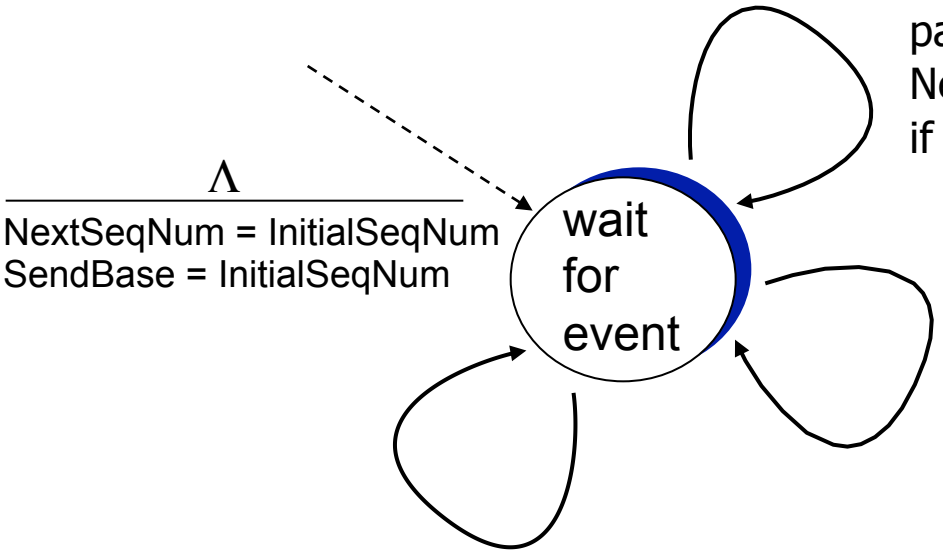
*timeout:*

- retransmit segment that caused timeout
- restart timer

*ack rcvd:*

- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments

# TCP sender (simplified)




data received from application above

create segment, seq. #: NextSeqNum
pass segment to IP (i.e., "send")
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

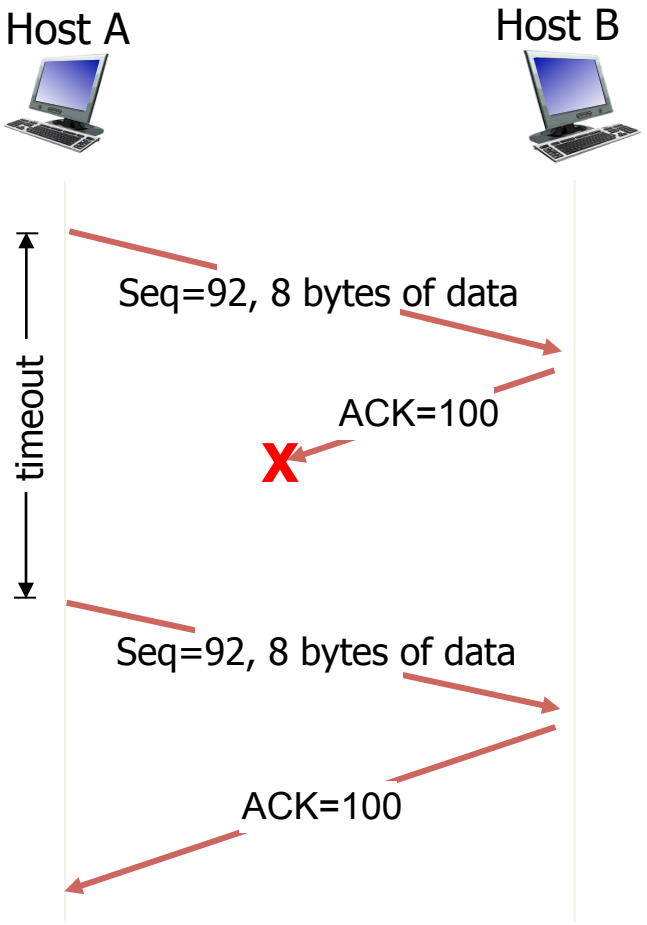
timeout

retransmit not-yet-acked segment
    with smallest seq. #
start timer

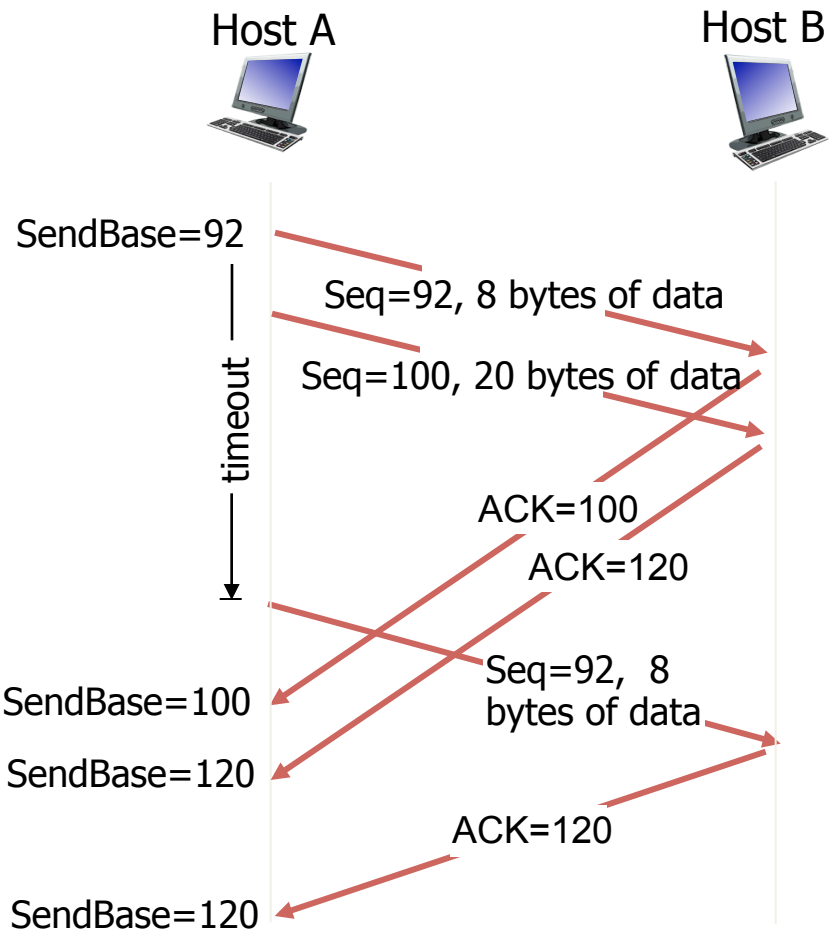
ACK received, with ACK field value y

```
if (y > SendBase) {
   SendBase = y
   /* SendBase–1: last cumulatively ACKed byte */
   if (there are currently not-yet-acked segments)
       start timer
     else stop timer
    }
```

# TCP: retransmission scenarios

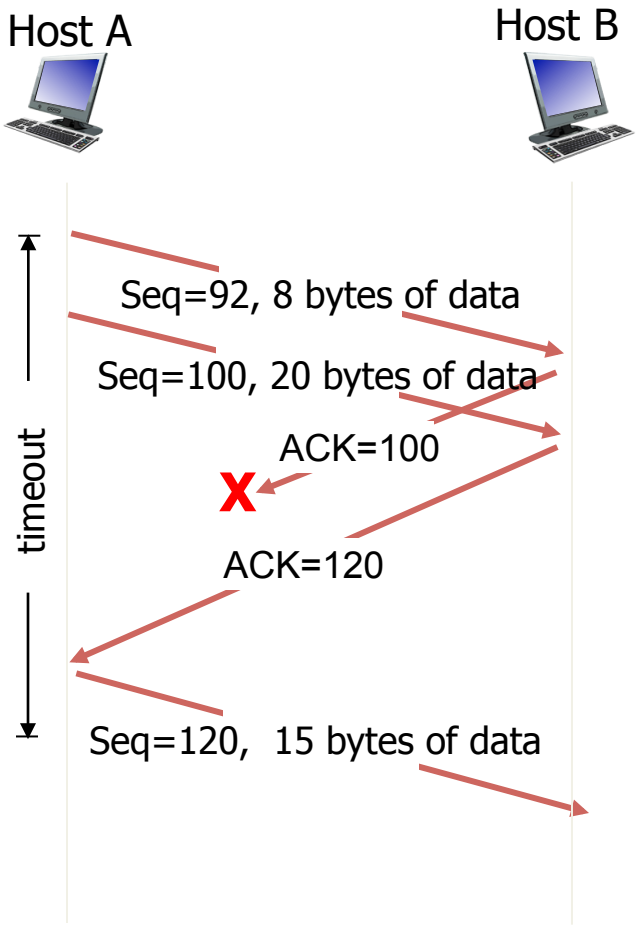



lost ACK scenario




premature timeout

# TCP: retransmission scenarios




cumulative ACK

# TCP ACK generation [RFC 1122, RFC 2581]

| *event at receiver* | *TCP receiver action* |
|---|---|
| arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed | delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK |
| arrival of in-order segment with expected seq #. One other segment has ACK pending | immediately send single cumulative ACK, ACKing both in-order segments |
| arrival of out-of-order segment higher-than-expect seq. # . Gap detected | immediately send *duplicate ACK*, indicating seq. # of next expected byte |
| arrival of segment that partially or completely fills gap | immediate send ACK, provided that segment starts at lower end of gap |

# TCP fast retransmit

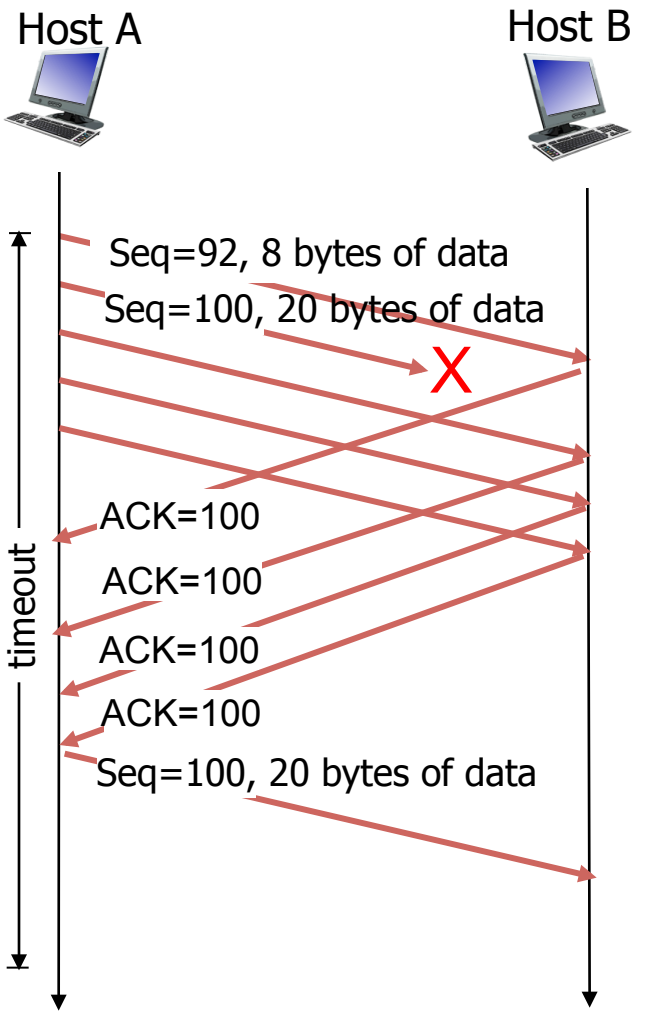
- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

*TCP fast retransmit*

if sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq #

- likely that unacked segment lost, so don't wait for timeout

# TCP fast retransmit




fast retransmit after sender receipt of triple duplicate ACK

# Differences between TCP reliable data transfer and Go-back-N

A. TCP only retransmits oldest (single) unacked segment, GBN retransmits all segments in the window

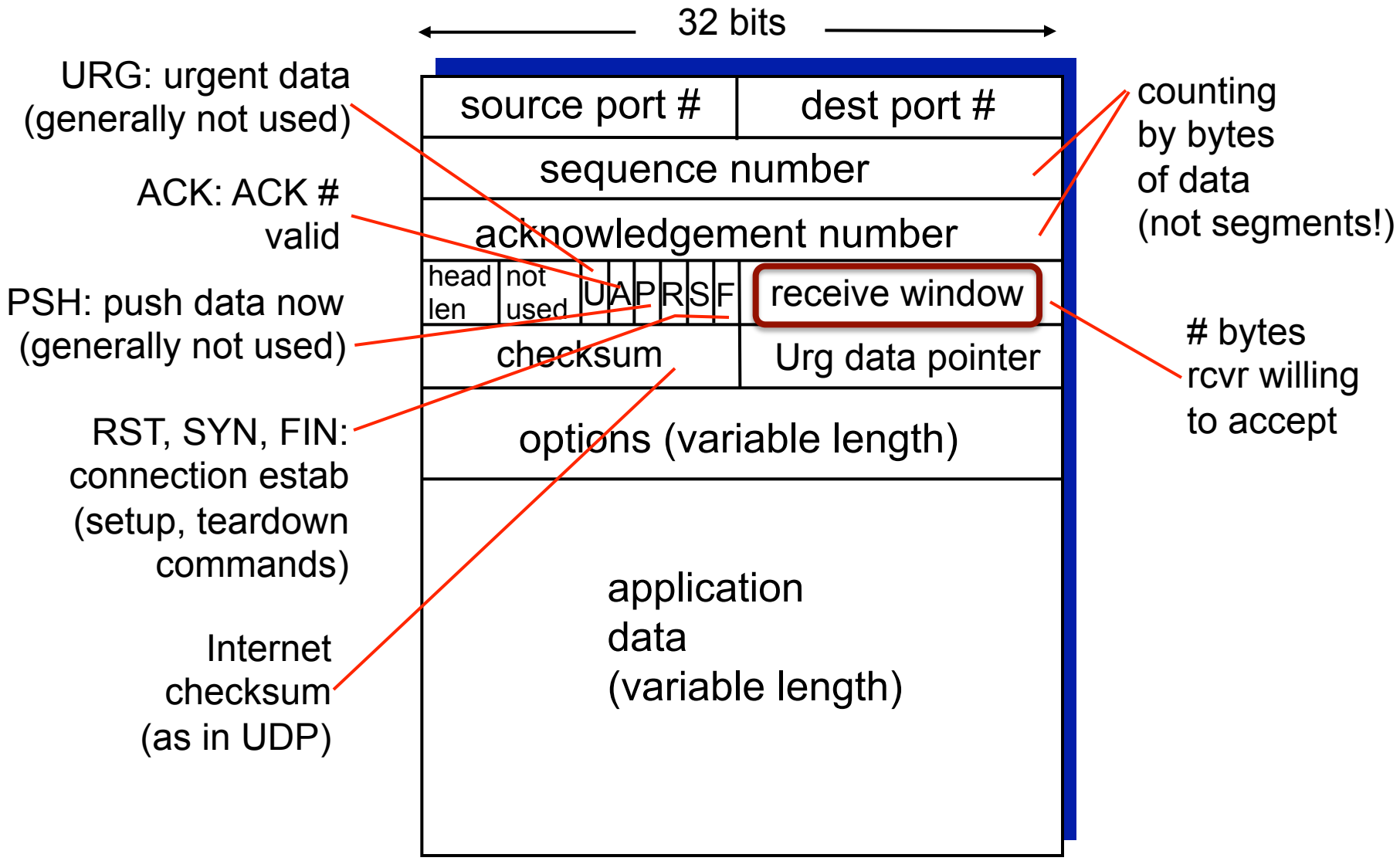
B. TCP uses duplicate ACK to retransmit, GBN does not

C. TCP uses cumulative ACK, GBN uses single ACK

D. A and B

E. A, B and C

# TCP segment structure



# What the “receive window” in a TCP segment header is used for?

A. Pipelining

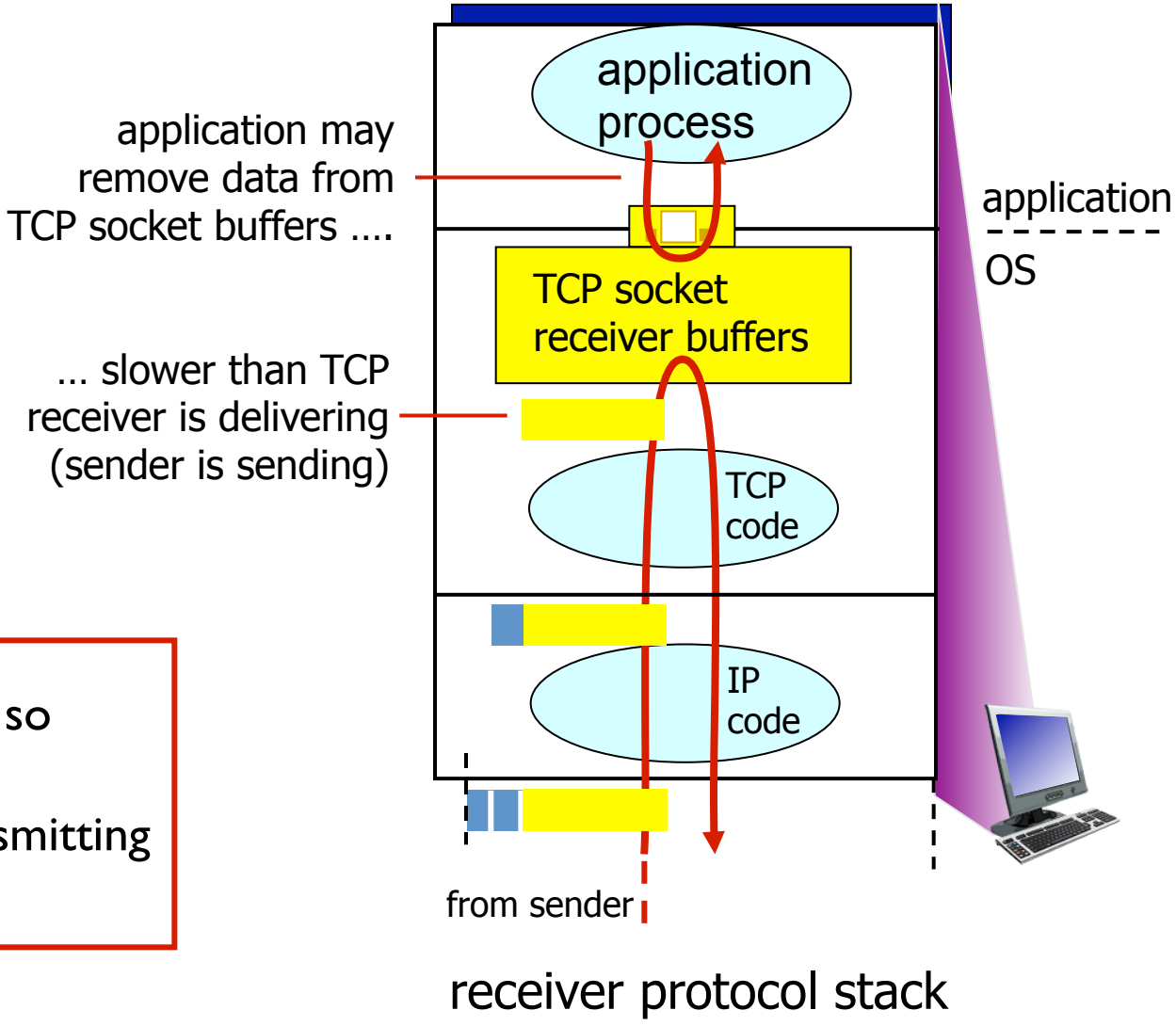
B. Size of window

C. Flow control

D. A and B

E. A, B, and C

# TCP flow control

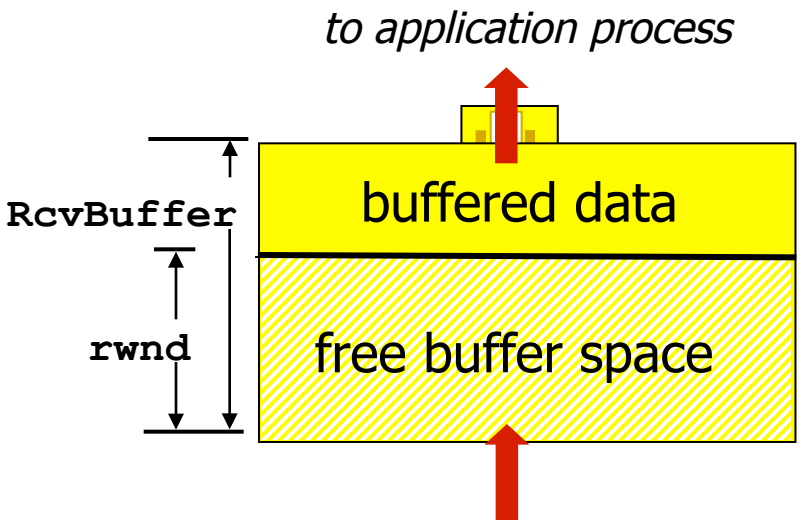



***flow control***

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

# TCP flow control

- receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



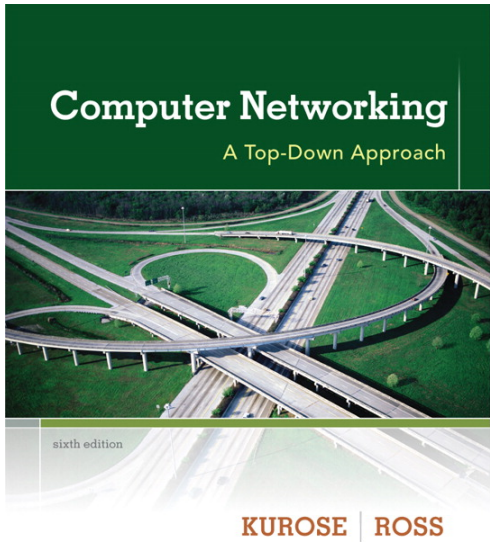

*receiver-side buffering*

# Next lecture

- Congestion control
  - Readings 3.6
- Guest lecture on Monday Feb 23rd
  - DNS Security
- iClickers questions review Friday Feb 27th
- Midterm review Wednesday March 4th
- Midterm exam in class
  - In class: 1 PM Friday, March 6th

# TCP joke ☺

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out.
...Hello, would you like to hear a TCP joke?"




Copy right notice:
These slides are adapted from J.F Kurose and K.W. Ross's ones



# *Computer Networking: A Top Down Approach*

6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012