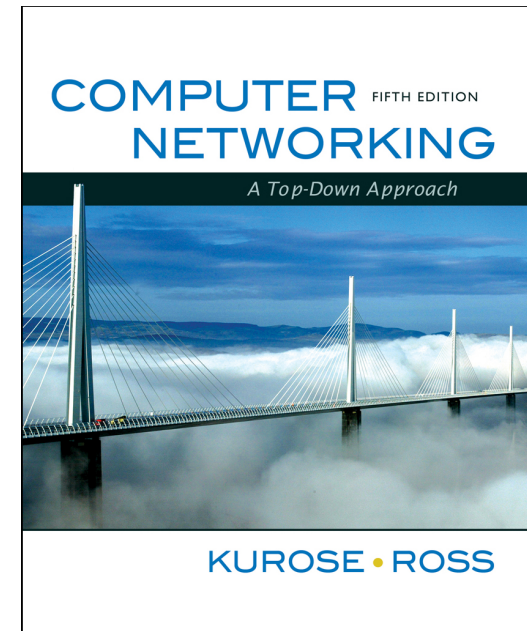


# Chapter 3

## Transport Layer



### A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- ❖ If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- ❖ If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2010  
J.F Kurose and K.W. Ross, All Rights Reserved

*Computer Networking:  
A Top Down Approach  
5<sup>th</sup> edition.  
Jim Kurose, Keith Ross  
Addison-Wesley, April  
2009.*

# Chapter 3 outline

3.1 Transport-layer services

3.2 Multiplexing and demultiplexing

3.3 Connectionless transport: UDP

3.4 Principles of reliable data transfer

- Bit error: Ack, seq.#
- Loss: Time out
- Pipelining
- Selective Repeat

3.5 Connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

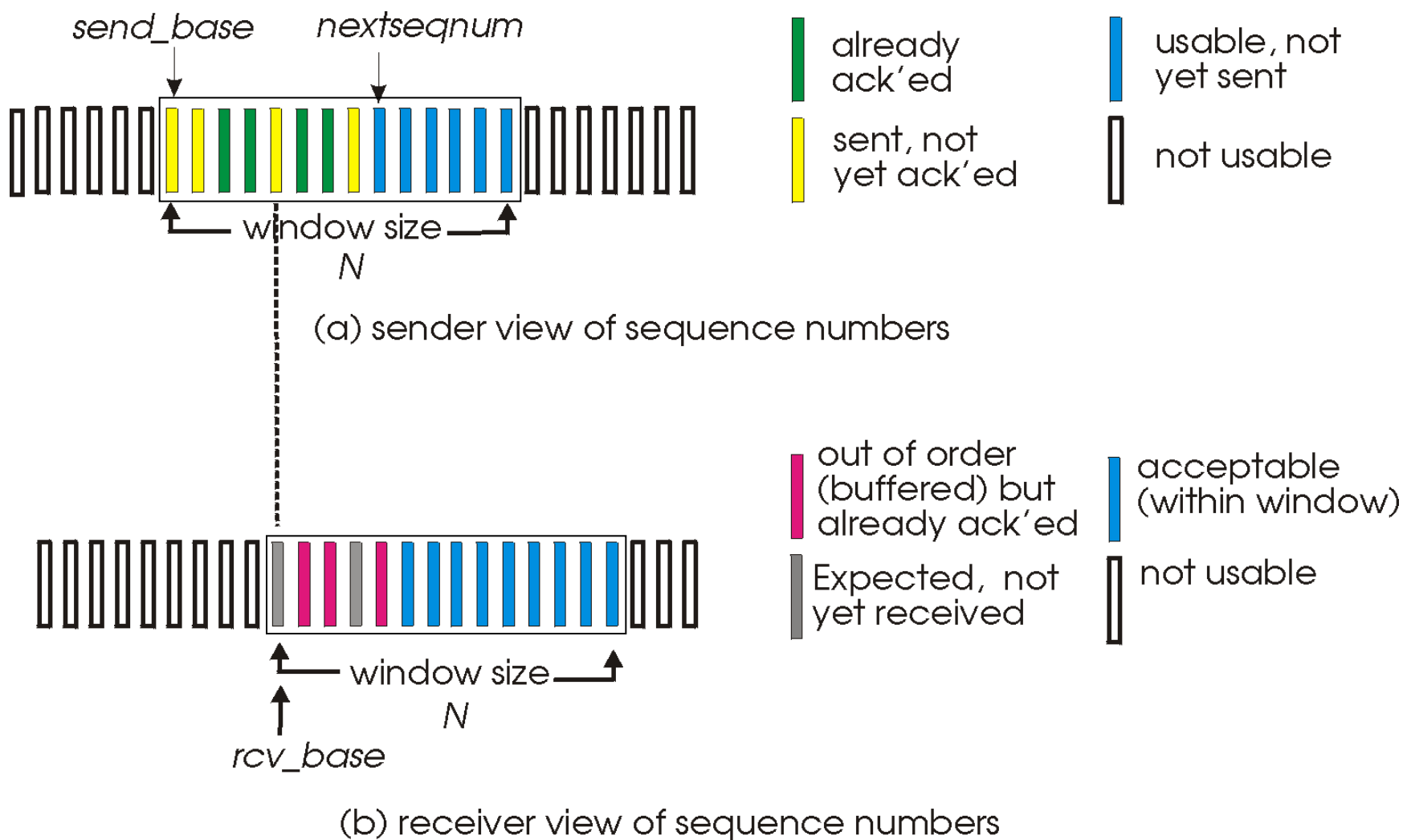
3.6 Principles of congestion control

3.7 TCP congestion control

# Selective Repeat

- ❖ receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- ❖ sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- ❖ sender window
  - N consecutive seq #'s
  - again limits seq #'s of sent, unACK'ed pkts

# Selective repeat: sender, receiver windows



# Selective repeat

## sender

### data from above :

- ❖ if next available seq # in window, send pkt

### timeout(n):

- ❖ resend pkt n, restart timer

### ACK(n) in [sendbase,sendbase+N]:

- ❖ mark pkt n as received
- ❖ if n smallest unACKed pkt, advance window base to next unACKed seq #

## receiver

### pkt n in [rcvbase,rcvbase+N-1]

- ❖ send ACK(n)
- ❖ out-of-order: buffer
- ❖ in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

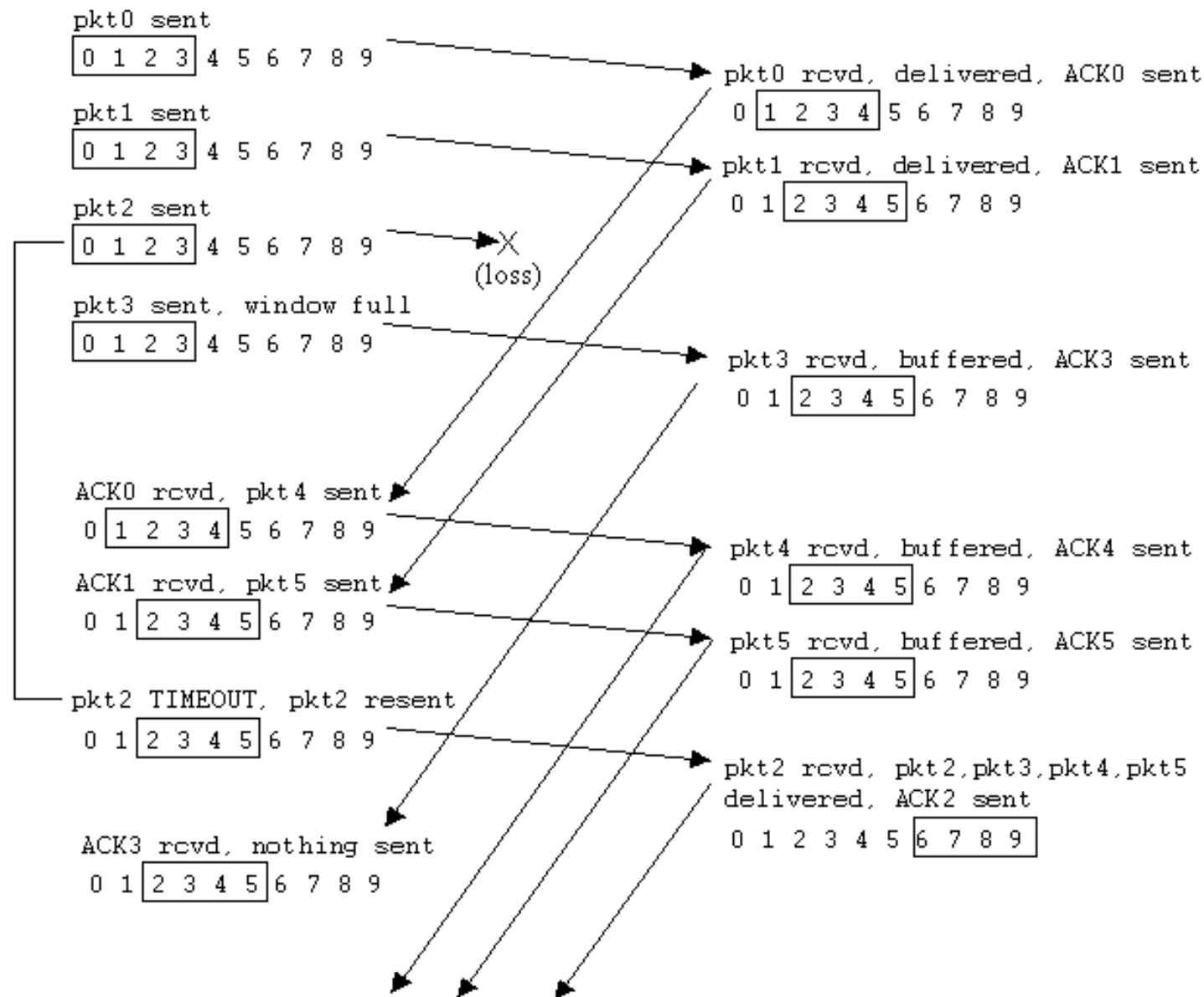
### pkt n in [rcvbase-N,rcvbase-1]

- ❖ ACK(n)

### otherwise:

- ❖ ignore

# Selective repeat in action



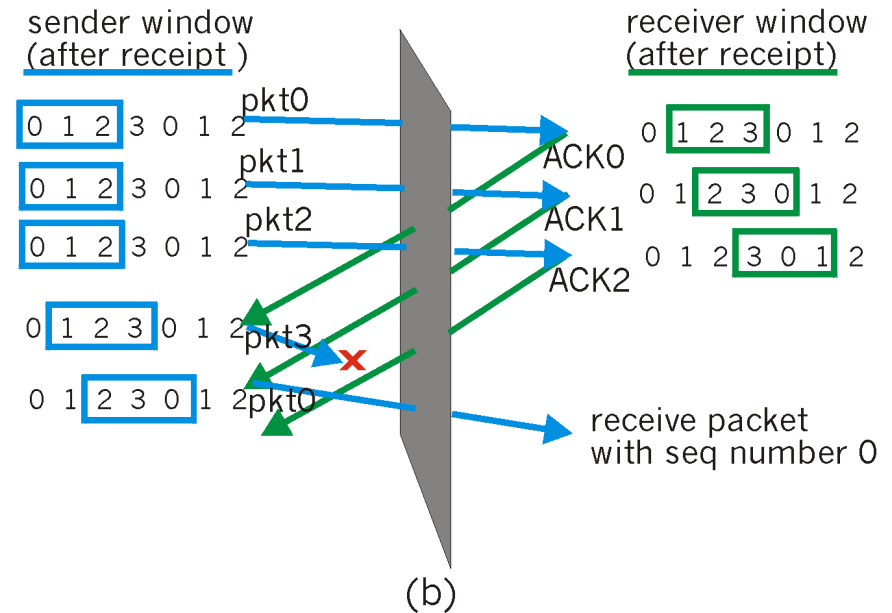
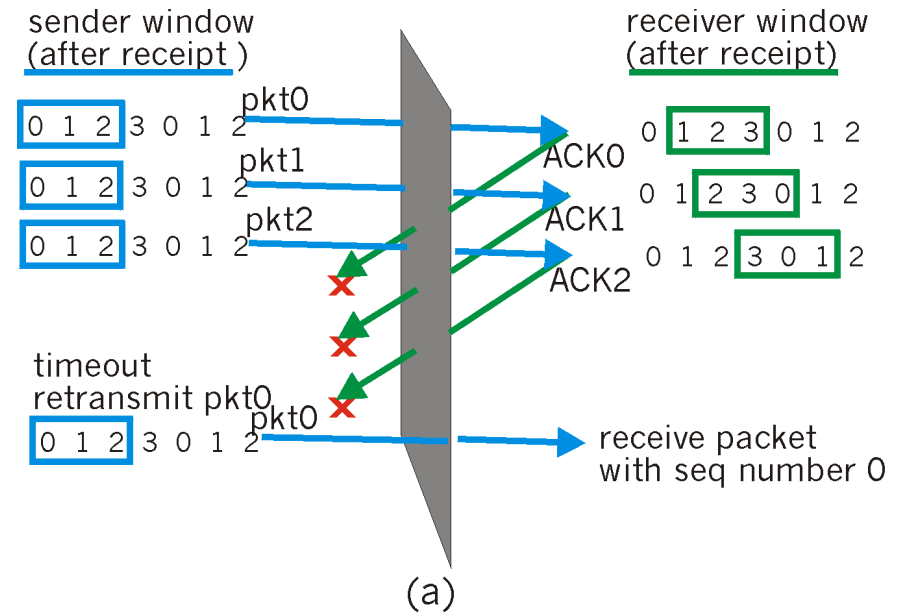
# Selective repeat: dilemma

Example:

- ❖ seq #'s: 0, 1, 2, 3
- ❖ window size=3

- ❖ receiver sees no difference in two scenarios!
- ❖ incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?



# Chapter 3 outline

3.1 Transport-layer services

3.2 Multiplexing and demultiplexing

3.3 Connectionless transport: UDP

3.4 Principles of reliable data transfer

3.5 Connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 Principles of congestion control

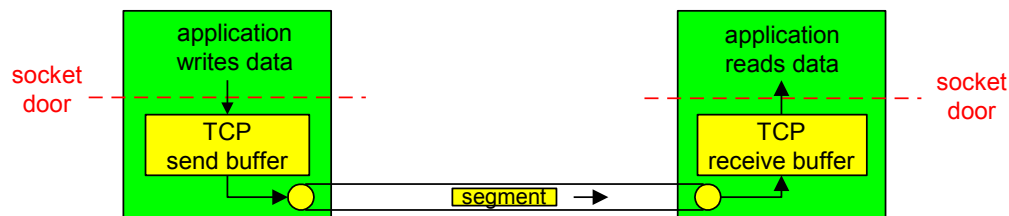
3.7 TCP congestion control



# TCP: Overview

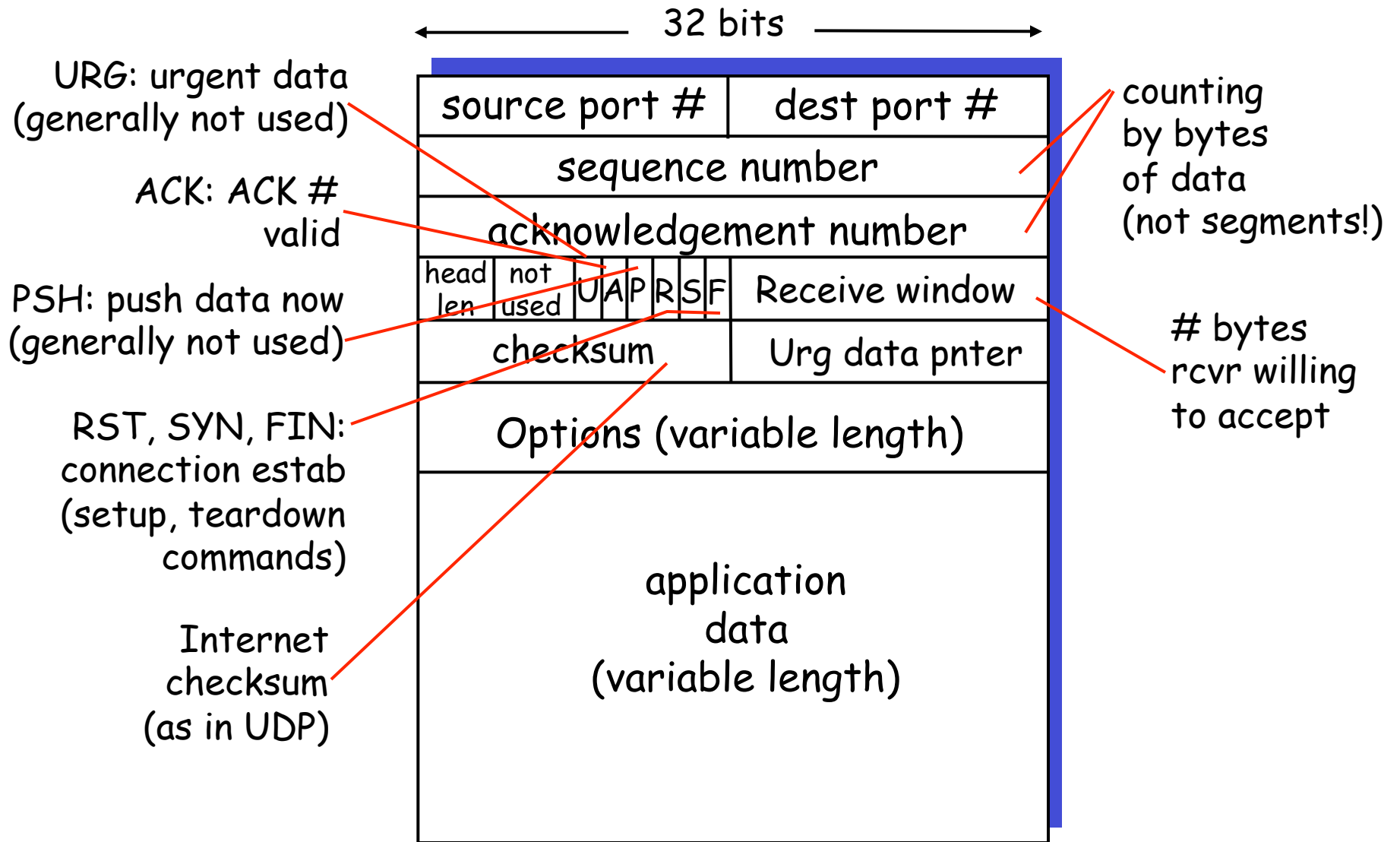
RFCs: 793, 1122, 1323, 2018, 2581

- ❖ **point-to-point:**
  - one sender, one receiver
- ❖ **reliable, in-order *byte stream*:**
  - no “message boundaries”
- ❖ **pipelined:**
  - TCP congestion and flow control set window size
- ❖ ***send & receive buffers***



- ❖ **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- ❖ **connection-oriented:**
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- ❖ **flow controlled:**
  - sender will not overwhelm receiver

# TCP segment structure



# TCP seq. #'s and ACKs

## Seq. #'s:

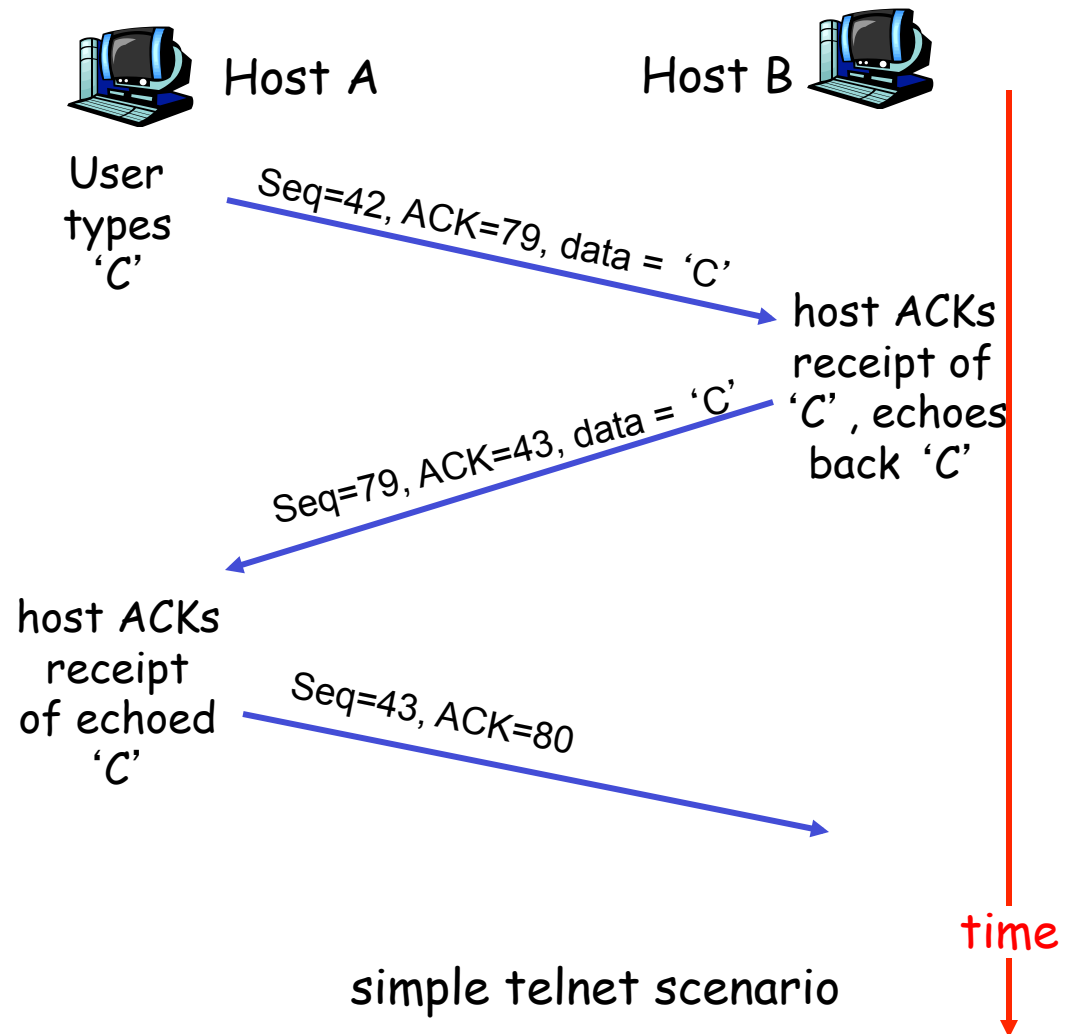
- byte stream  
"number" of first byte in segment's data

## ACKs:

- seq # of next byte expected from other side
- cumulative ACK

**Q:** how receiver handles out-of-order segments

- A:** TCP spec doesn't say, - up to implementor



# TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?

- ❖ longer than RTT
  - but RTT varies
- ❖ too short: premature timeout
  - unnecessary retransmissions
- ❖ too long: slow reaction to segment loss

Q: how to estimate RTT?

- ❖ **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- ❖ **SampleRTT** will vary, want estimated RTT “smoother”
  - average several recent measurements, not just current **SampleRTT**

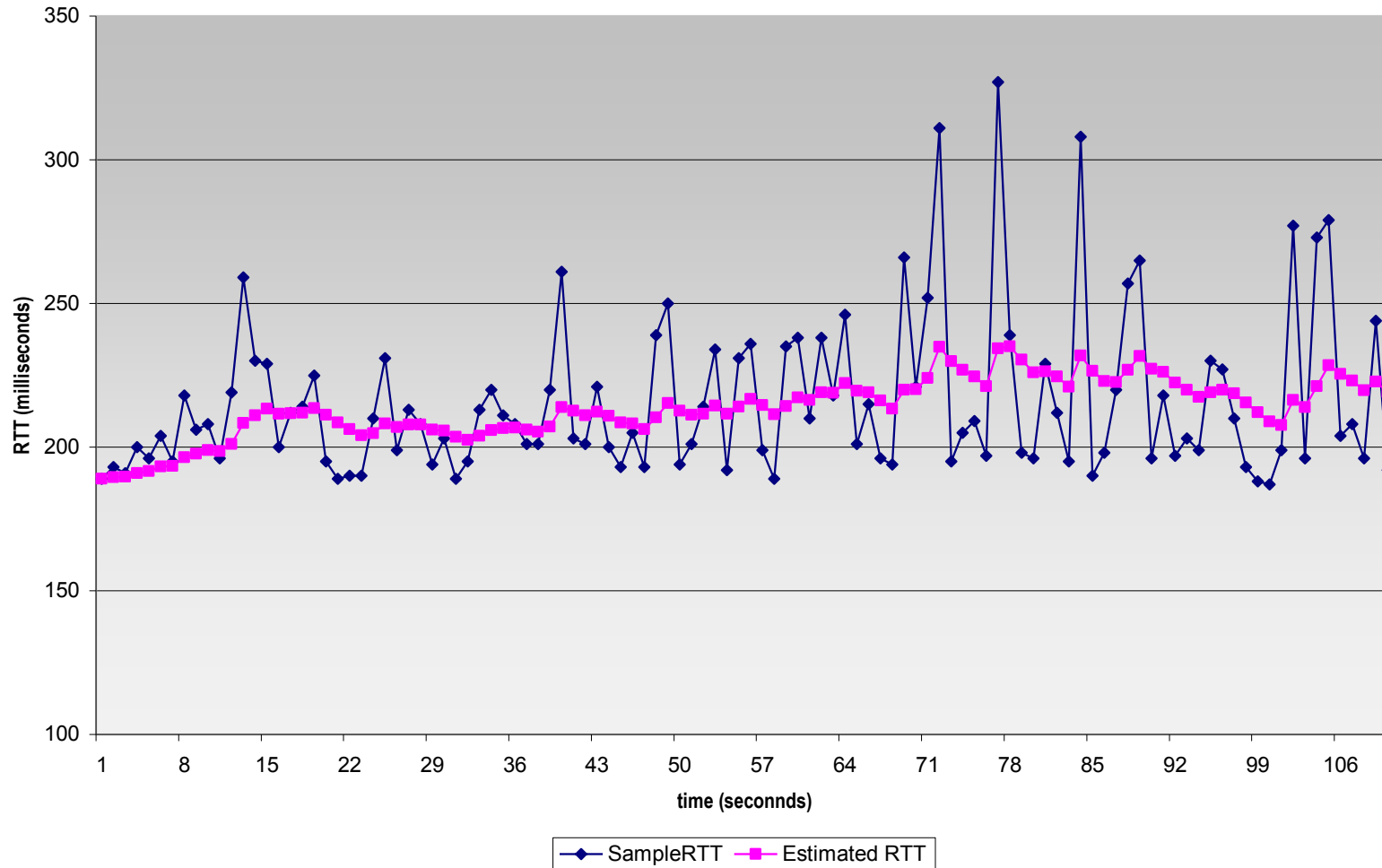
# TCP Round Trip Time and Timeout

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- ❖ Exponential weighted moving average
- ❖ influence of past sample decreases exponentially fast
- ❖ typical value:  $\alpha = 0.125$

# Example RTT estimation:

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



# TCP Round Trip Time and Timeout

## Setting the timeout

- ❖ EstimatedRTT plus “safety margin”
  - large variation in EstimatedRTT → larger safety margin
- ❖ first estimate of how much SampleRTT deviates from EstimatedRTT:

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically,  $\beta = 0.25$ )

Then set timeout interval:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$