National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 13

Chapter 3

6th October, 2022

Nauman Moazzam Hayat

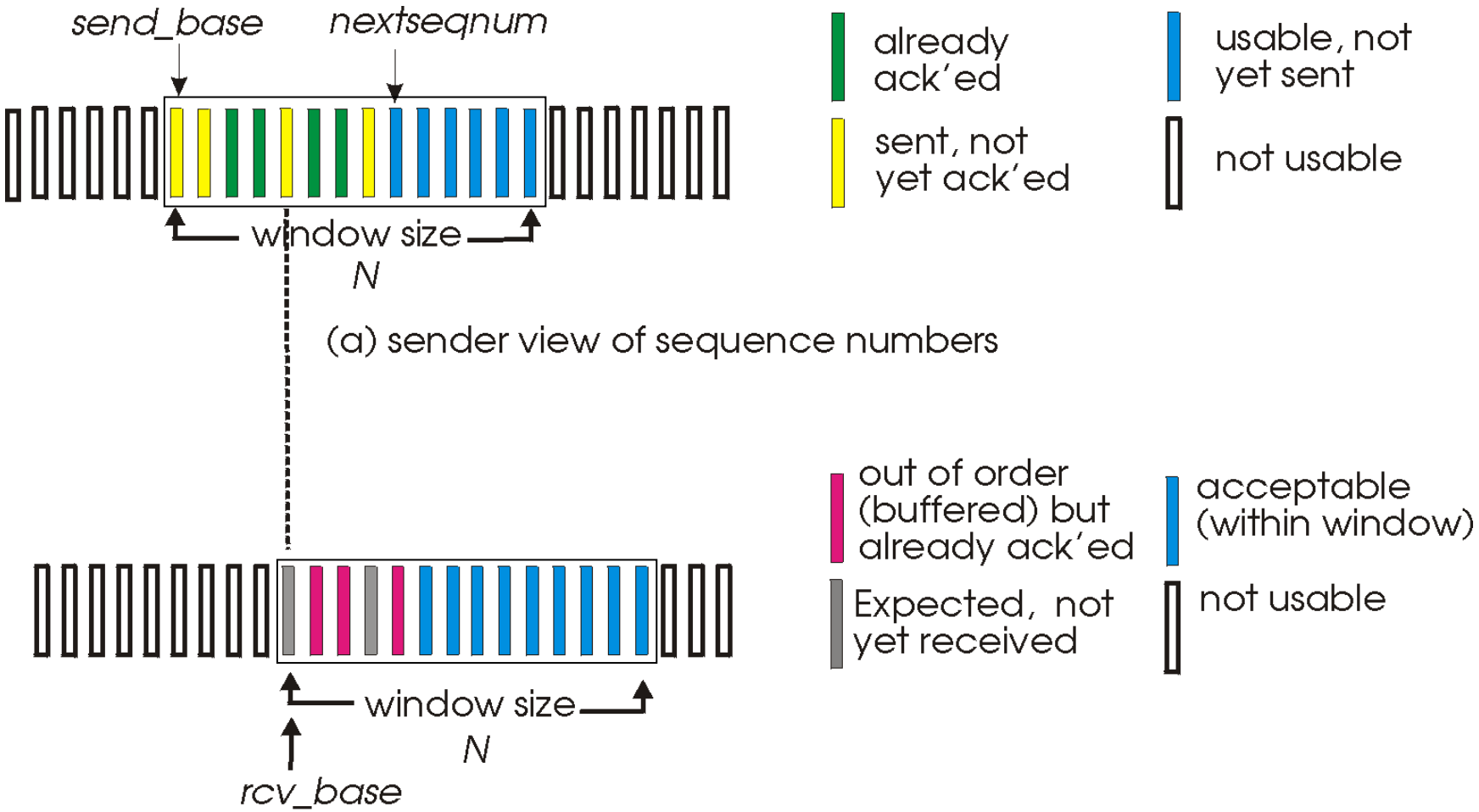
nauman.moazzam@lhr.nu.edu.pk

Office Hours: 02:30 pm till 06:00 pm (Every Tuesday & Thursday)

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
  - limits seq #s of sent, unACKed pkts
- Sender window size = Receiver window size =N

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

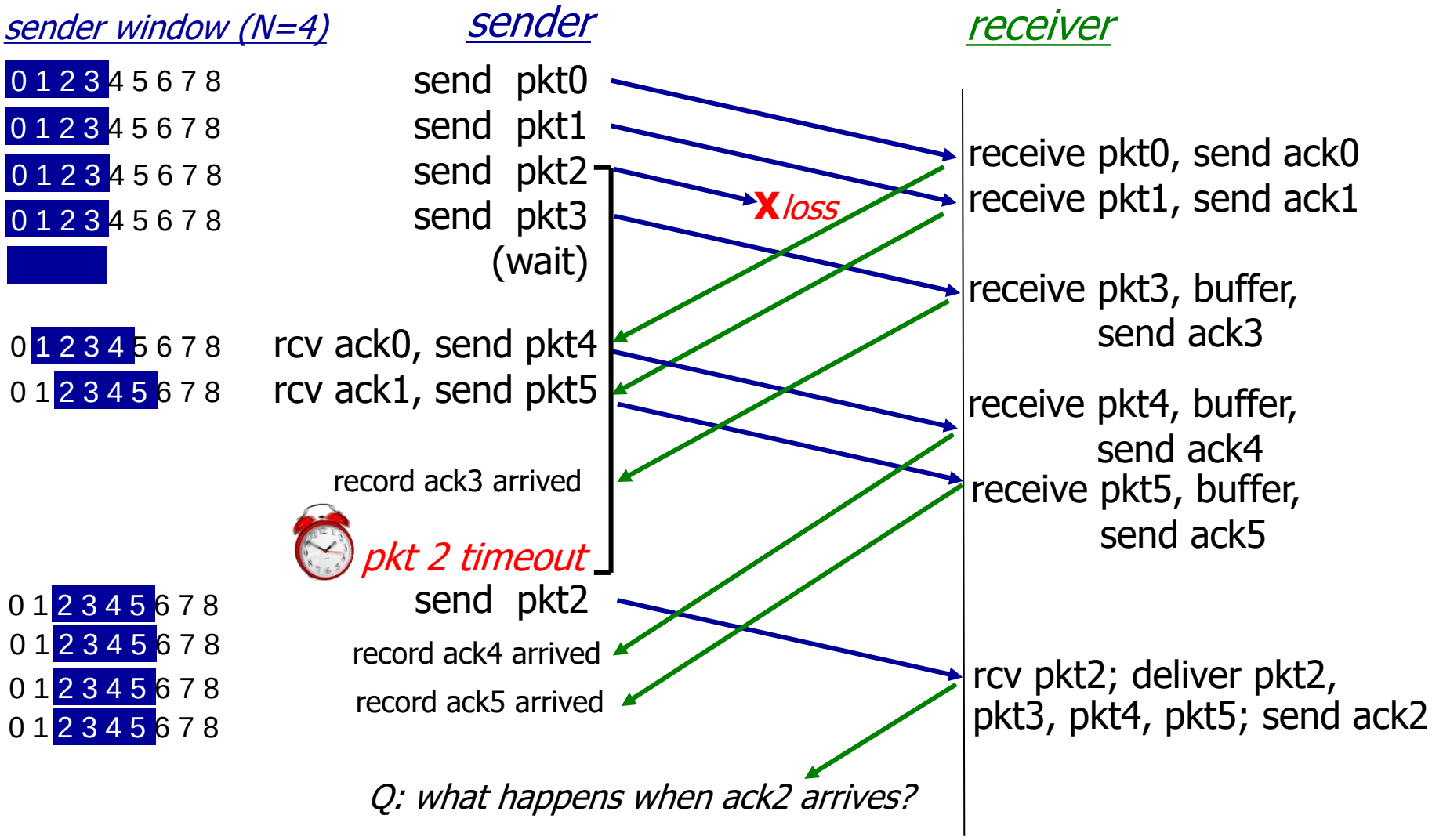
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective repeat in action



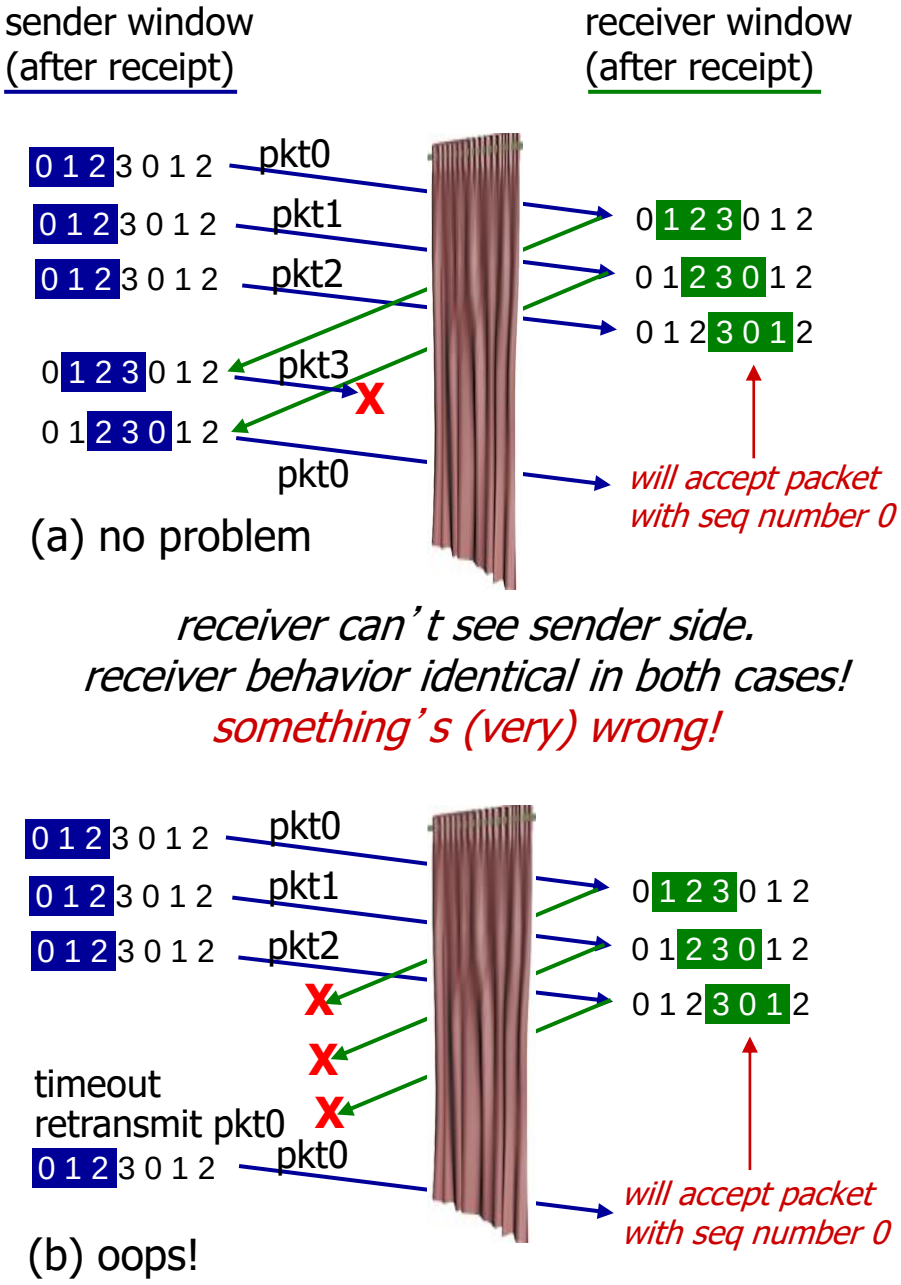
Selective repeat: dilemma

example:

- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b) (new packet or retransmission?)

Q: what relationship between seq # size and window size to avoid problem in (b)?

Window size should be less than or equal to half the sequence number space in SR protocol. This is to avoid packets being recognized incorrectly. If the windows size is greater than half the sequence number space, then if an ACK is lost, the sender may send new packets that the receiver believes are retransmissions.



Major Differences between Stop & Wait, Go-Back-N (GBN) and Selective Repeat (SR) [1/2]

- *Stop and Wait*

The sender sends the packet and waits for the ACK (acknowledgement) of the packet. Once the ACK reaches the sender, it transmits the next packet in row. If the ACK is not received, it re-transmits the previous packet again.

- *Go Back N*

The sender sends N packets which is equal to the window size. Once the entire window is sent, the sender then waits for a cumulative ACK to send more packets. On the receiver end, it receives only in-order packets and discards out-of-order packets. As in case of packet loss, the entire window would be re-transmitted.

- *Selective Repeat*

The sender sends packet of window size N and the receiver acknowledges all packet whether they were received in order or not. In this case, the receiver maintains a buffer to contain out-of-order packets and sorts them. The sender selectively re-transmits the lost packet and moves the window forward.

URL for Interactive Animations:

https://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198702.cw/index.html

Differences between Stop & Wait, Go-Back-N (GBN) and Selective Repeat (SR) [2/2]

PROPERTIES	STOP AND WAIT	GO BACK N	SELECTIVE REPEAT
Sender window size	1	N	N
Receiver Window size	1	1	N
Minimum Sequence number	2	N+1	2N
Efficiency	1/(1+2*a)	N/(1+2*a)	N/(1+2*a)
Type of Acknowledgement	Individual	Cumulative	Individual
Supported order at Receiving end	–	In-order delivery only	Out-of-order delivery as well
Number of retransmissions in case of packet drop	1	N	1

Where,

- ***a = Ratio of Propagation delay and Transmission delay***
- ***At N=1, Go Back N is effectively reduced to Stop and Wait***
- ***As Go Back N acknowledges the packets cumulatively, it rejects out-of-order packets***
- ***As Selective Repeat supports receiving out-of-order packets (it sorts the window after receiving the packets), it uses Independent Acknowledgement to acknowledge the packets.***

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

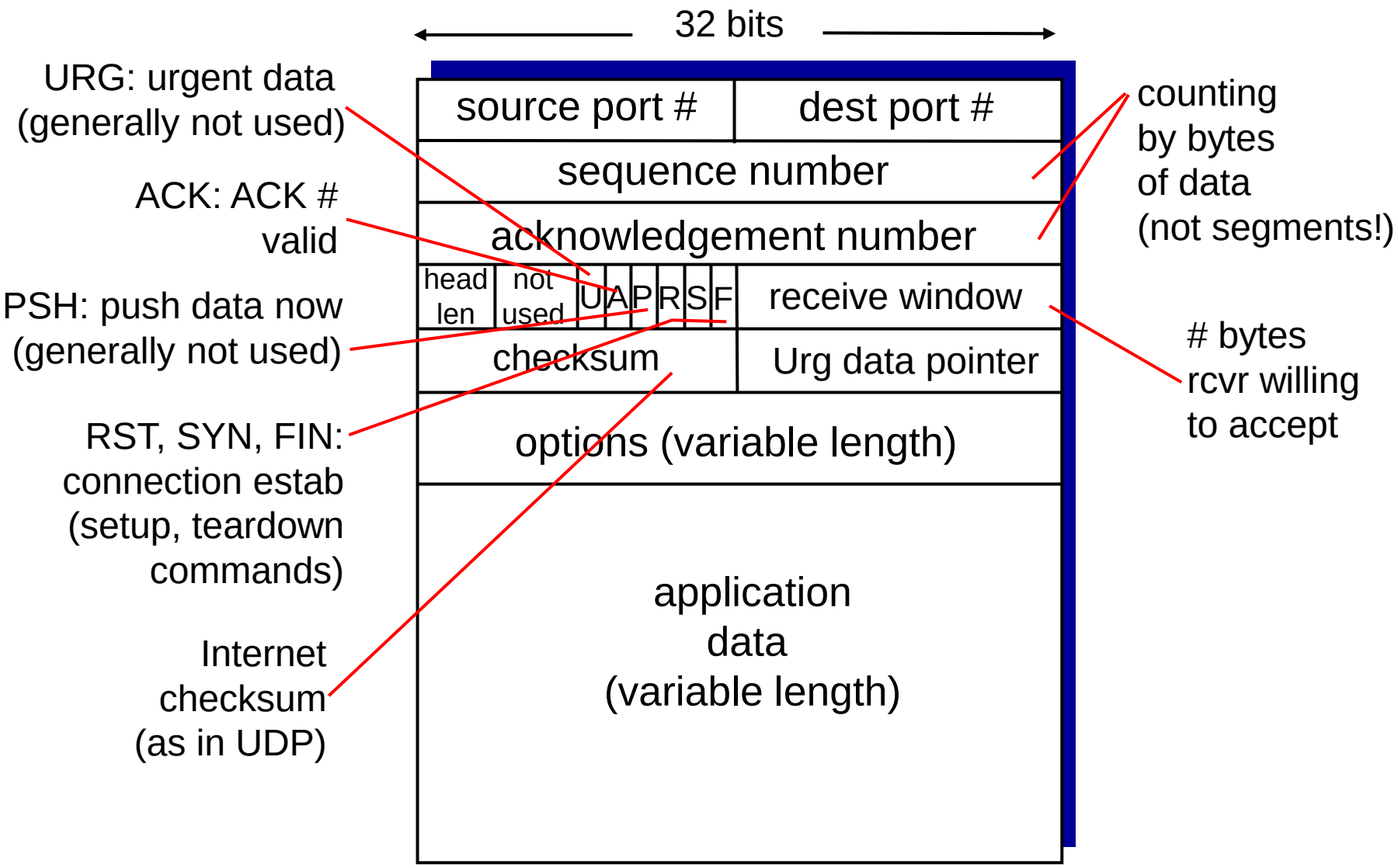
3.6 principles of congestion control

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
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size. Is based on Path MTU. (It is the max. amount of app layer data in the segment, not the max size of TCP segment including the TCP header.)
- connection-oriented:
  - 3-way handshaking (exchange of control msgs) initiates sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



Difference between PUSH & Urgent Flags in TCP Header

PSH	URG
--> All data in buffer to be pushed to NL(sender)/ AL(receiver).	--> Only the urgent data to be given to AL immediately.
--> Data is delivered in sequence.	--> Data is delivered out of sequence.

(marked by the Urgent Data Pointer field.)

TCP seq. numbers, ACKs

sequence numbers:

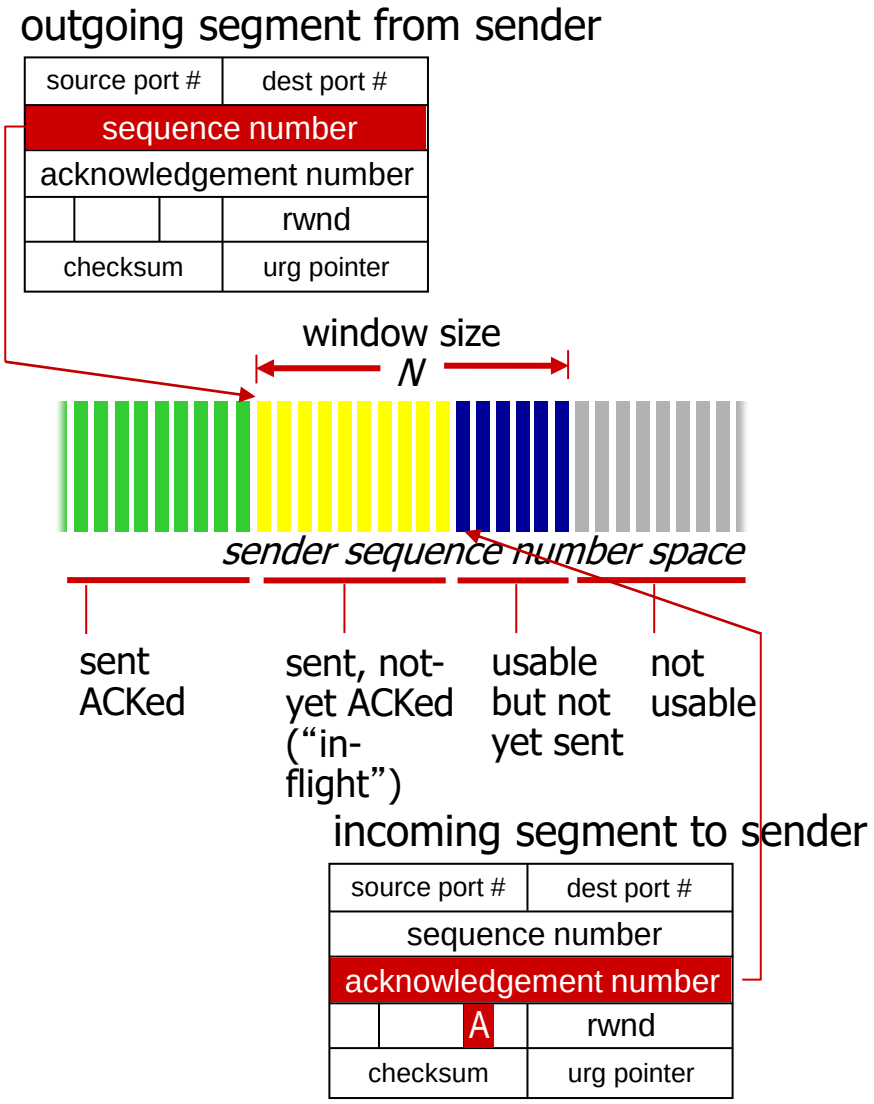
- byte stream "number" of first byte in segment's data

acknowledgements:

- 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 (two basic choices, i.e. either discard out-of-order segment or buffer it.)



Example of Sequence Number & ACK Number

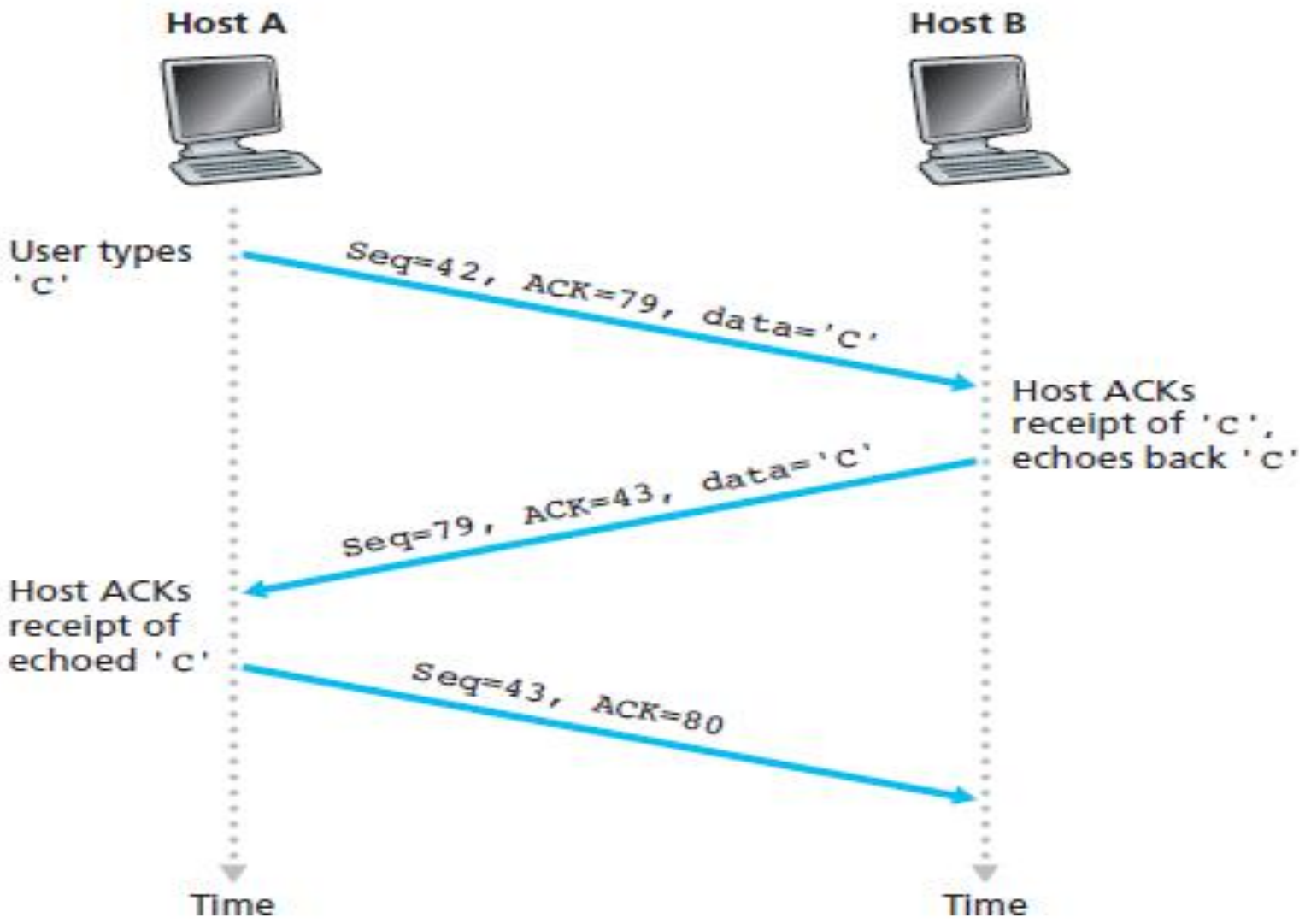


Figure 3.31 ♦ Sequence and acknowledgment numbers for a simple Telnet application over TCP

TCP round trip time, 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

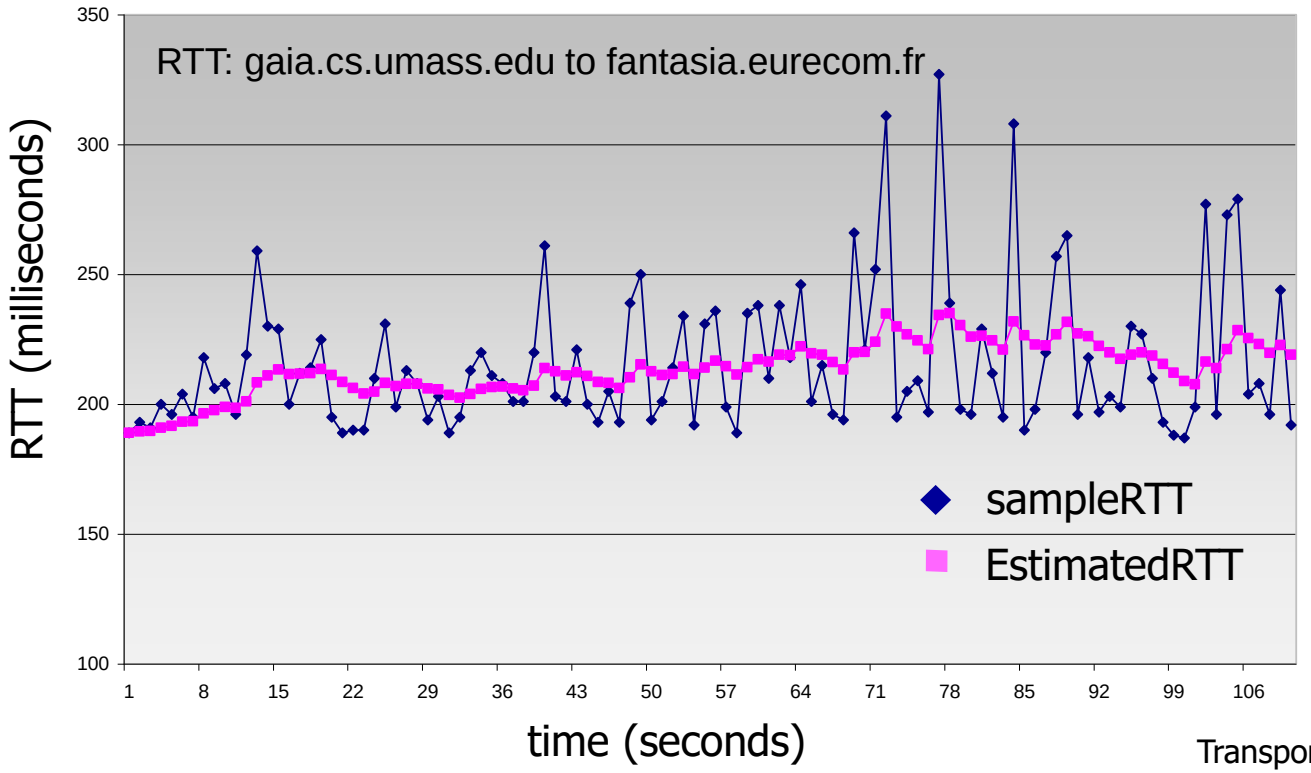
Q: how to estimate RTT?

- **`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, timeout

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125 (i.e. 1/8)



TCP round trip time, timeout

- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT +
                β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)
```

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

"safety margin"

Doubling the Timeout Interval

This modification provides a limited form of congestion control

- In this modification, whenever the timeout event occurs, each time TCP retransmits, it sets the next timeout interval to twice the previous value, rather than deriving it from the last EstimatedRTT and DevRTT
- Thus the intervals grow exponentially after each retransmission
- However, whenever the timer is started after either of the two other events (that is, data received from application above and/or ACK received), the TimeoutInterval is derived from the most recent values of EstimatedRTT and DevRTT