DATA COMMUNICATOIN NETWORKING

Instructor: Ouldooz Baghban Karimi

Course Book: Computer Networking, A Top-Down Approach
By: Kurose, Ross

Course Overview

- **Basics of Computer Networks**
  - Internet & Protocol Stack
  - Application Layer
  - Transport Layer
  - Network Layer
  - Data Link Layer
- **Advanced Topics**
  - Case Studies of Computer Networks
  - Internet Applications
  - Network Management
  - Network Security

Pipelined Protocols

Go-back-N:

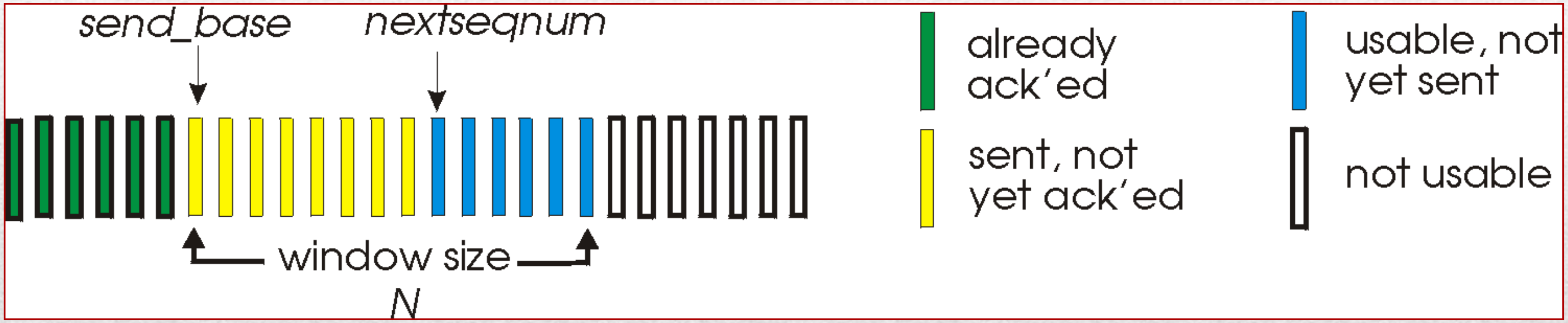
- Sender can have up to N un-ACKed packets in pipeline
- Receiver only sends *cumulative ACK*
  - Does not ACK packet if there is a gap
- Sender has timer for oldest un-ACKed packet
  - When timer expires, retransmit *all* un-ACKed packets

Selective Repeat:

- Sender can have up to N un-ACKed packets in pipeline
- Receiver sends *individual ACKs* for each packet
- Sender maintains timer for each un-ACKed packet
  - When timer expires, retransmit only that un-ACKed packet

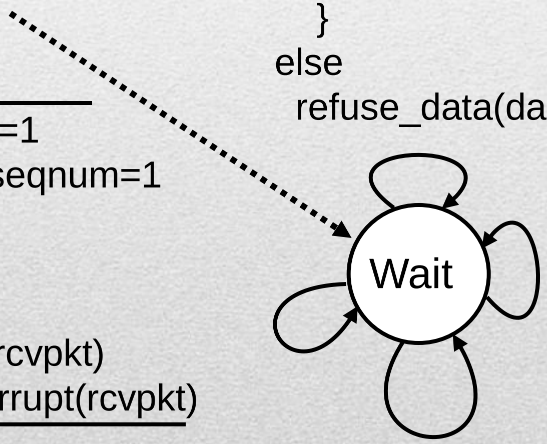
Go-Back-N: Sender

- k-bit sequence number in packet header
- "window" of up to N, consecutive unACKed packets allowed



- ACK(n): ACKs all packets up to, including n - *cumulative ACK*
  - May receive duplicate ACKs
  - Timer for oldest in-flight packet
- *Timeout(n)*: retransmit packet n and all higher sequence number packets in window

GBN: Sender FSM



```
rdt_send(data)

if (nextseqnum < base+N) {
    sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
    udt_send(sndpkt[nextseqnum])
    if (base == nextseqnum)
       start_timer
    nextseqnum++
    }
else
  refuse_data(data)
```

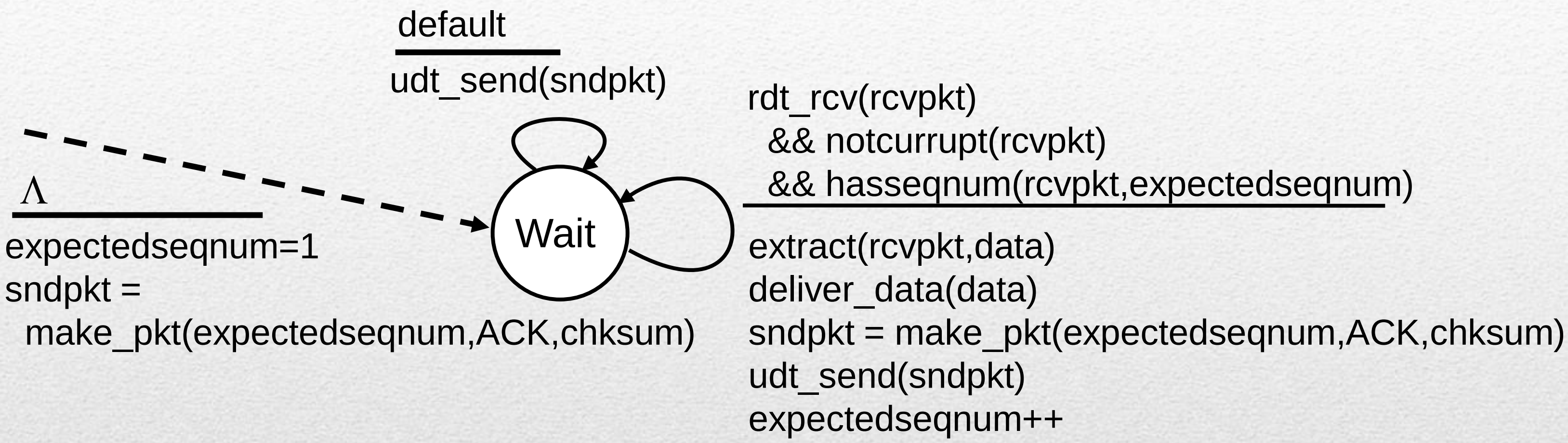
```
Λ

base=1
nextseqnum=1
```

```
timeout

start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
…
udt_send(sndpkt[nextseqnum-1])
```

```
rdt_rcv(rcvpkt)
   && corrupt(rcvpkt)
```

```
rdt_rcv(rcvpkt) &&
   notcorrupt(rcvpkt)

base = getacknum(rcvpkt)+1
If (base == nextseqnum) stop_timer else start_timer
```

GBN: Receiver FSM



default

udt_send(sndpkt)

Λ

expectedseqnum=1
sndpkt =
 make_pkt(expectedseqnum,ACK,chksum)

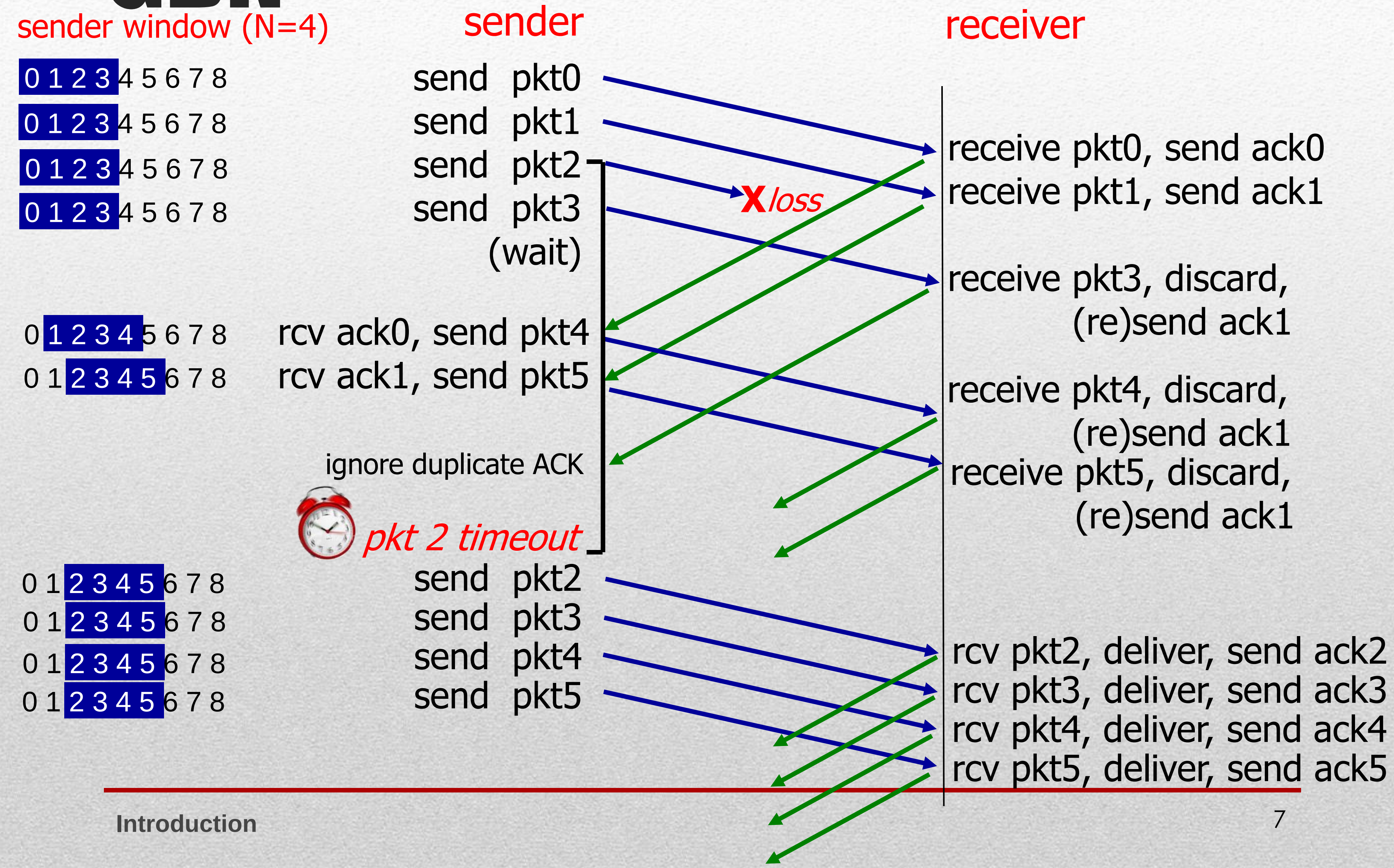
Wait

rdt_rcv(rcvpkt)
 && notcurrupt(rcvpkt)
 && hasseqnum(rcvpkt,expectedseqnum)

extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(expectedseqnum,ACK,chksum)
udt_send(sndpkt)
expectedseqnum++

- ACK-only: always send ACK for correctly-received packet with highest in-order sequence number
  - May generate duplicate ACKs
  - Need only remember **`expectedseqnum`**
- Out-of-order packet
  - Discard (do not buffer): no receiver buffering!
  - Re-ACK packet with highest in-order sequence number

GBN



sender window (N=4)

sender

receiver

0 1 2 3 4 5 6 7 8 send pkt0

0 1 2 3 4 5 6 7 8 send pkt1

0 1 2 3 4 5 6 7 8 send pkt2

0 1 2 3 4 5 6 7 8 send pkt3

(wait)

X loss

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, discard,
(re)send ack1

0 1 2 3 4 5 6 7 8 rcv ack0, send pkt4

0 1 2 3 4 5 6 7 8 rcv ack1, send pkt5

receive pkt4, discard,
(re)send ack1

ignore duplicate ACK

receive pkt5, discard,
(re)send ack1

pkt 2 timeout

0 1 2 3 4 5 6 7 8 send pkt2

0 1 2 3 4 5 6 7 8 send pkt3

0 1 2 3 4 5 6 7 8 send pkt4

0 1 2 3 4 5 6 7 8 send pkt5

rcv pkt2, deliver, send ack2

rcv pkt3, deliver, send ack3

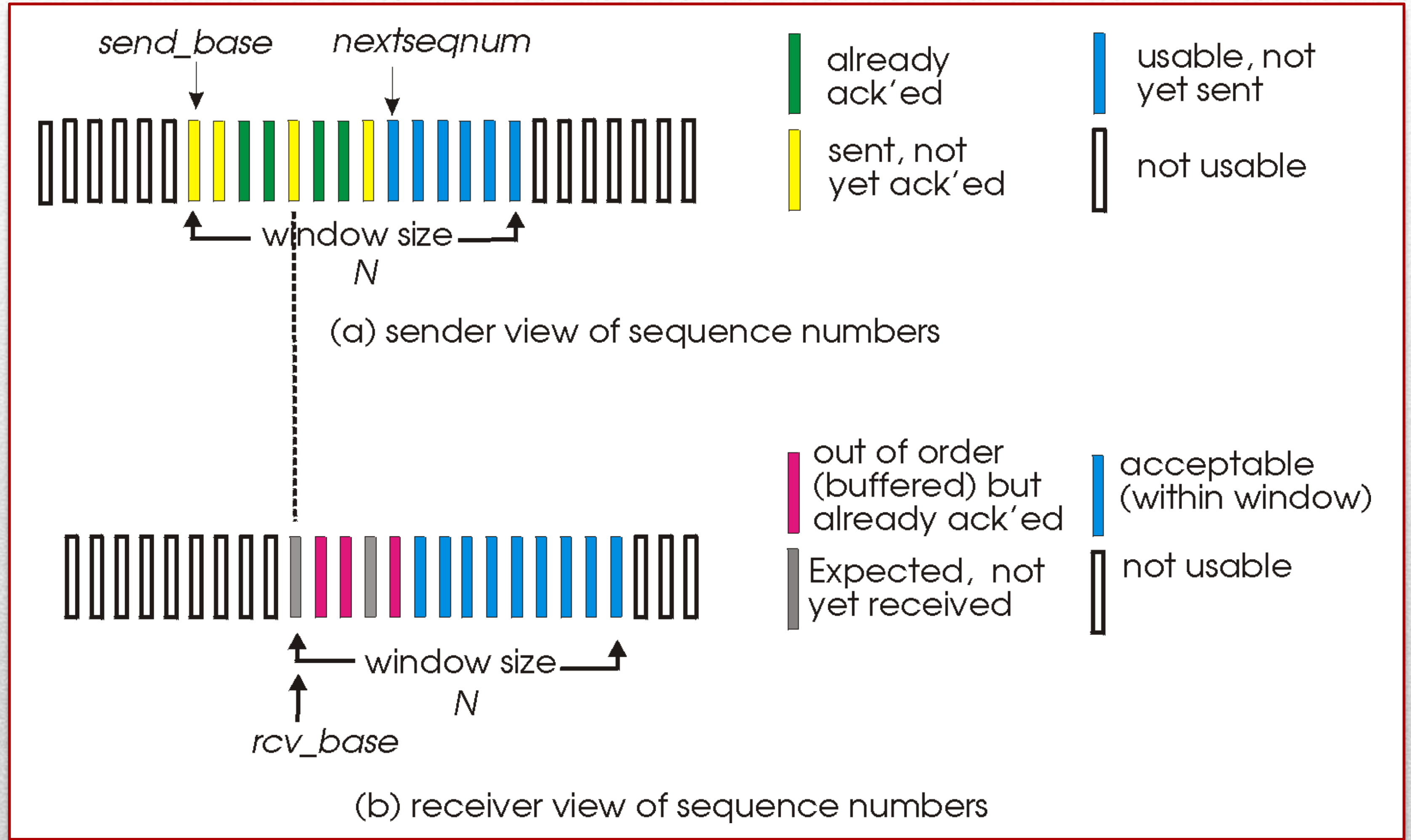
rcv pkt4, deliver, send ack4

rcv pkt5, deliver, send ack5

Selective Repeat

- **Receiver acknowledges every correctly received packets**
  - Buffers packets, as needed, for eventual in-order delivery to upper layer
- **Sender only resends packets for which ACK not received**
  - Sender timer for each unACKed packet
- **Sender window**
  - N consecutive sequence numbers
  - Limits sequence numbers of sent, unACKed packets

Selective Repeat: Windows


send_base
nextseqnum
already ack'ed
usable, not yet sent
sent, not yet ack'ed
not usable
window size N
(a) sender view of sequence numbers
out of order (buffered) but already ack'ed
acceptable (within window)
Expected, not yet received
not usable
window size N
rcv_base
(b) receiver view of sequence numbers

Selective Repeat

Sender

Data from above:

- if next available sequence number in window, send packet

Timeout(n):

- Resend packet n, restart timer

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

- Mark packet n as received
- If n smallest unACKed pkt, advance window base to next unACKed sequence number

Receiver

packet n in [rcvbase, rcvbase+N-1]

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

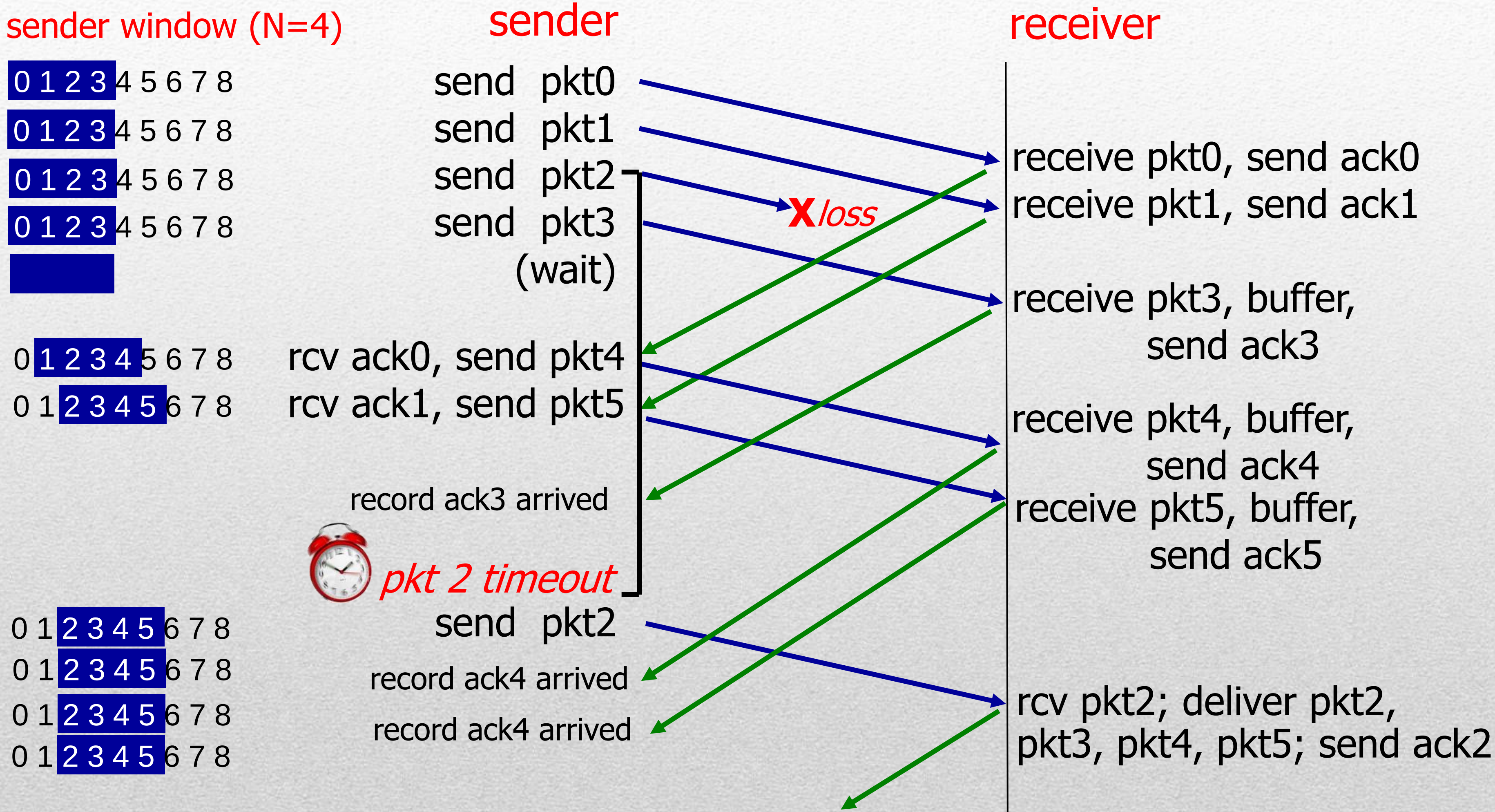
Packet n in [rcvbase-N,rcvbase-1]

- ACK(n)

Otherwise:

- Ignore

Selective Repeat



sender window (N=4)

sender

receiver

0 1 2 3 4 5 6 7 8 — send pkt0 — receive pkt0, send ack0

0 1 2 3 4 5 6 7 8 — send pkt1 — receive pkt1, send ack1

0 1 2 3 4 5 6 7 8 — send pkt2 — X loss

0 1 2 3 4 5 6 7 8 — send pkt3 — receive pkt3, buffer, send ack3

(wait)

0 **1 2 3 4** 5 6 7 8 — rcv ack0, send pkt4 — receive pkt4, buffer, send ack4

0 1 **2 3 4 5** 6 7 8 — rcv ack1, send pkt5 — receive pkt5, buffer, send ack5

record ack3 arrived

pkt 2 timeout

0 1 **2 3 4 5** 6 7 8 — send pkt2 — rcv pkt2; deliver pkt2, pkt3, pkt4, pkt5; send ack2

0 1 **2 3 4 5** 6 7 8 — record ack4 arrived

0 1 **2 3 4 5** 6 7 8 — record ack4 arrived

0 1 **2 3 4 5** 6 7 8

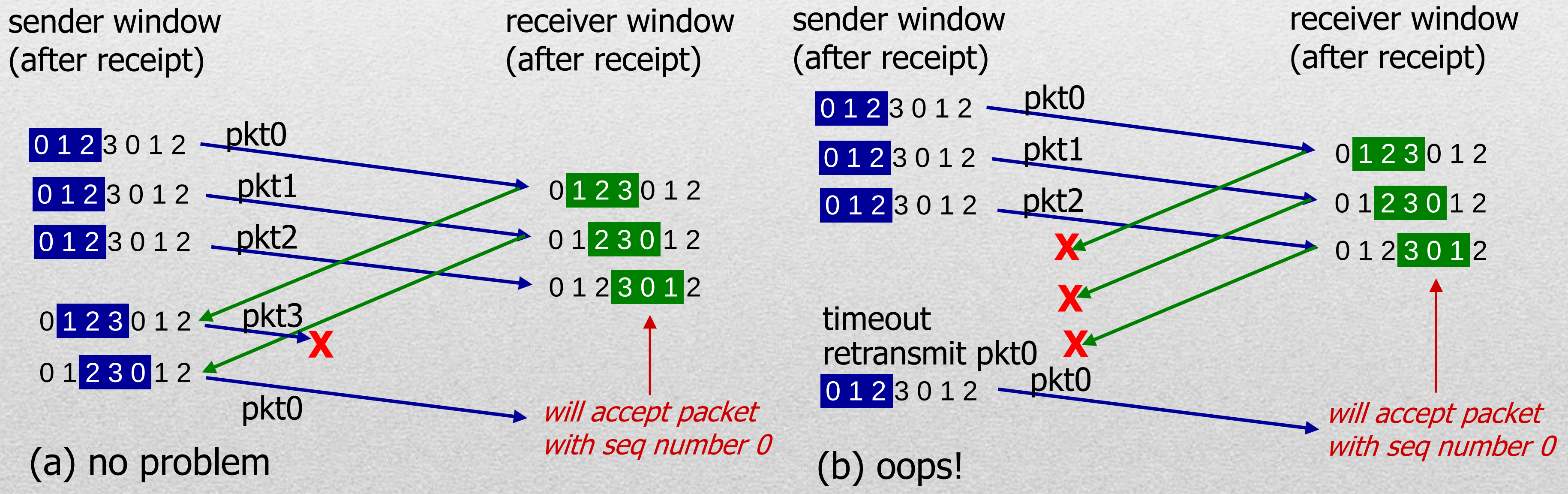
Q: what happens when ack2 arrives?

Selective Repeat Dilemma

Example:

- Sequence number's: 0, 1, 2, 3
- Window size=3
- Receiver sees no difference in two scenarios!
- Duplicate data accepted as new in (b)

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



sender window (after receipt)
receiver window (after receipt)
0 1 2 3 0 1 2
pkt0
pkt1
pkt2
pkt3
X
pkt0
will accept packet with seq number 0
(a) no problem
sender window (after receipt)
receiver window (after receipt)
pkt0
pkt1
pkt2
X
X
X
timeout retransmit pkt0
pkt0
will accept packet with seq number 0
(b) oops!

Receiver can't see sender side. receiver behavior identical in both cases! Something (very) wrong!

TCP Overview

- RFCs: 793,1122,1323, 2018, 2581Application Layer
- **Point-to-point**
  - One sender
  - One receiver
- **Reliable, in-order byte steam**
  - No “message boundaries”
- **Pipelined**
- **Full duplex data**
  - Bi-directional data flow in same connection
  - MSS: maximum segment size
- **Connection-oriented**
  - Handshaking (exchange of control messages) initiates sender, receiver state before data exchange
- **Flow controlled**
  - Sender will not overwhelm receiver

TCP Overview

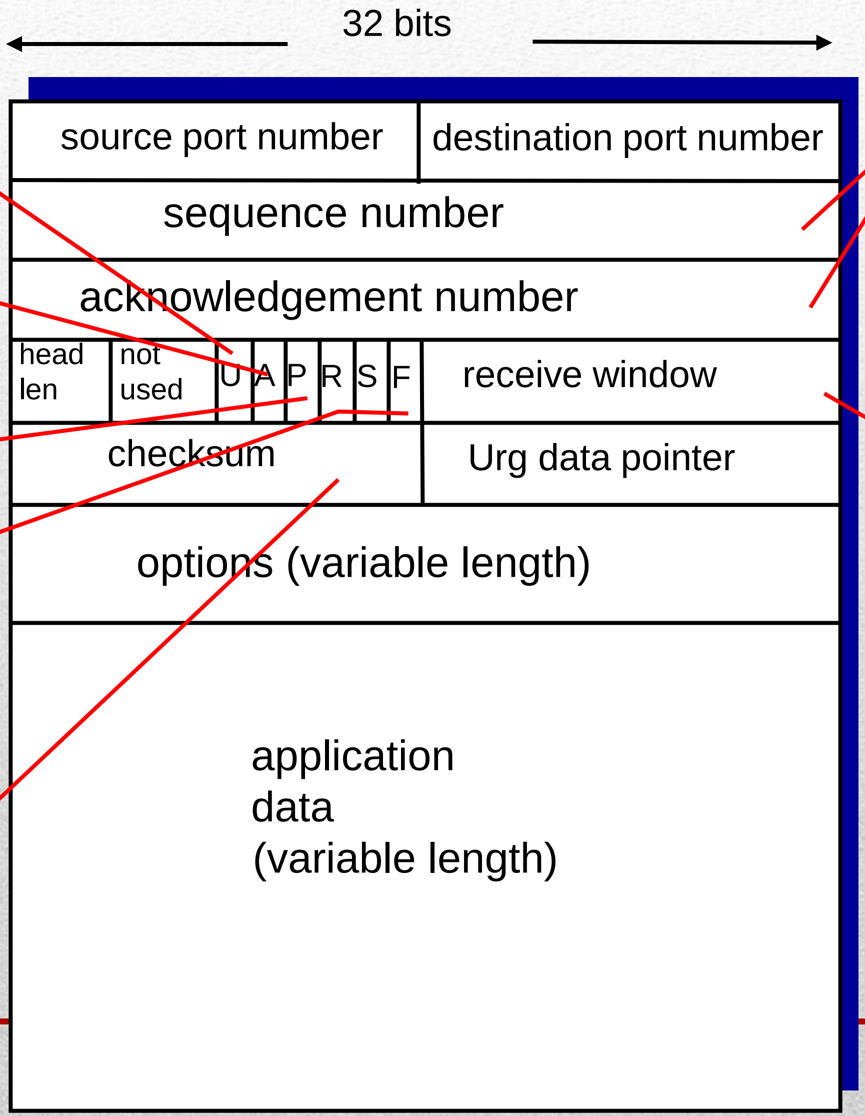
URG: urgent data (generally not used)

ACK: ACK number valid

PSH: push data now (generally not used)

RST, SYN, FIN: connection estab (setup, teardown commands)

Internet checksum (as in UDP)



counting by bytes of data (not segments!)

Number of bytes receiver willing to accept

TCP Sequence Numbers & ACKs

Sequence numbers

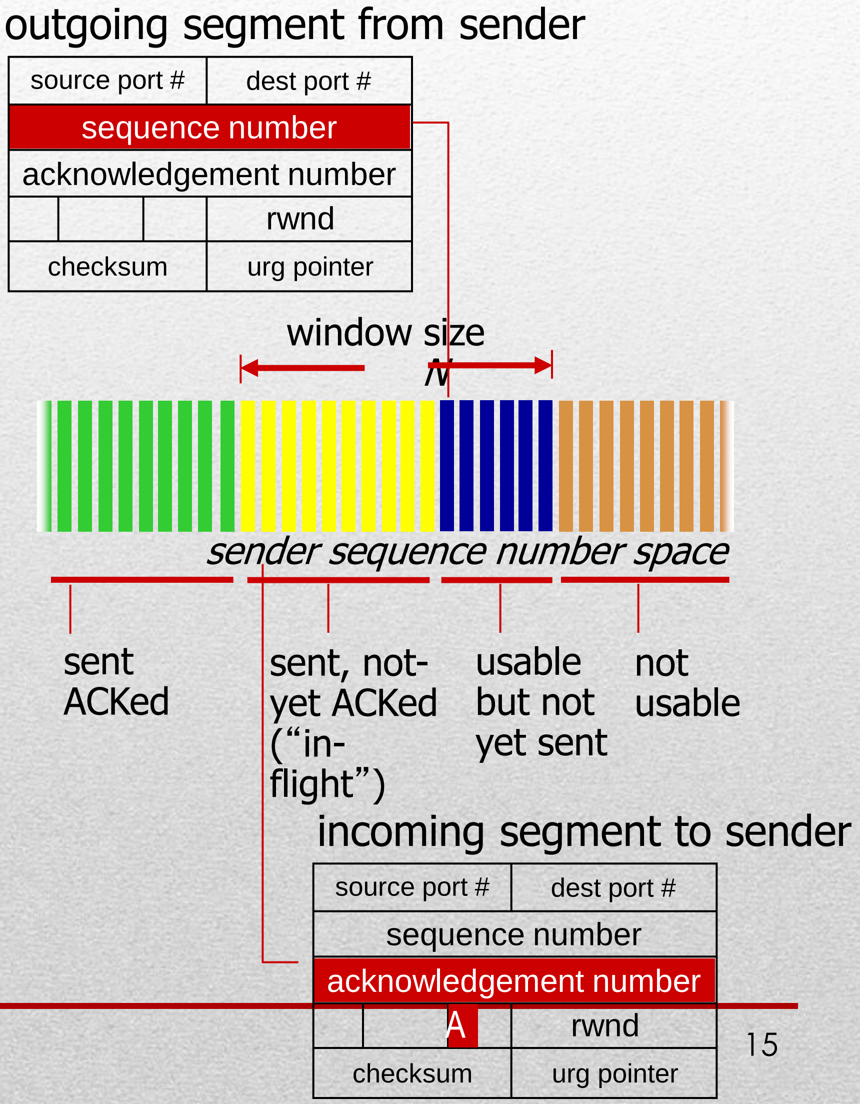
- Byte stream "number" of first byte in segment's data

Acknowledgements

- Sequence number of next byte expected from other side
- Cumulative ACK

Q: How receiver handles out-of-order segments?

A: TCP spec does not say



outgoing segment from sender

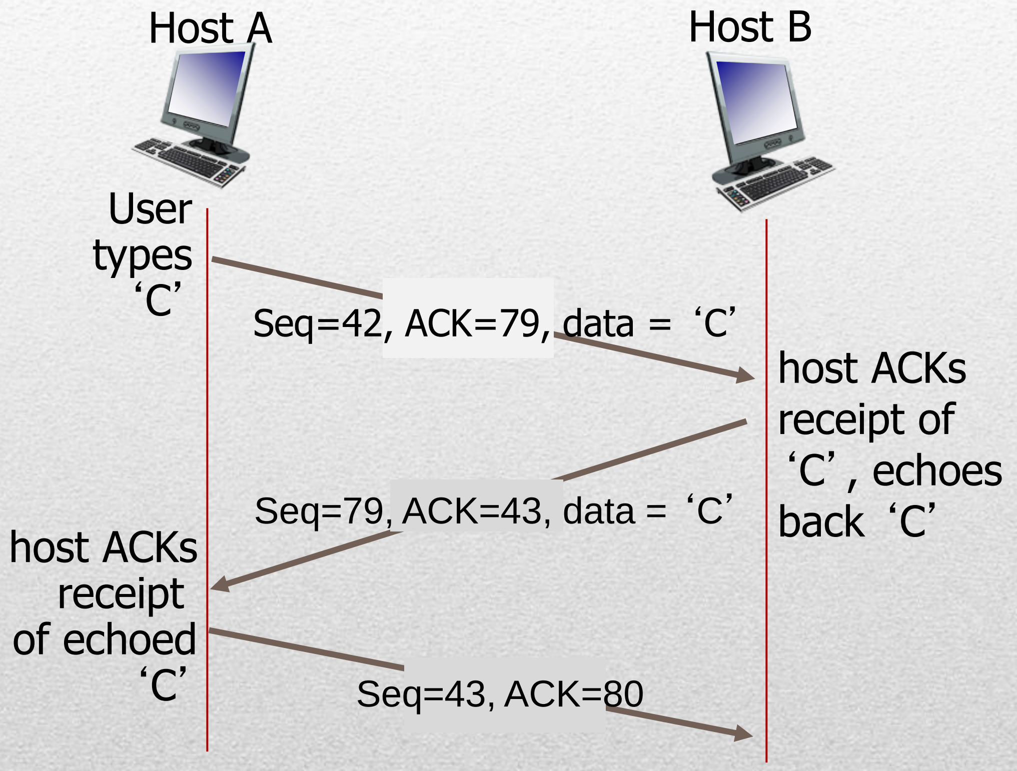
source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer

window size
N
sender sequence number space
sent ACKed
sent, not-yet ACKed ("in-flight")
usable but not yet sent
not usable

incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
A	rwnd
checksum	urg pointer

TCP Sequence Numbers & ACKs


Host A
Host B
User types ‘C’
Seq=42, ACK=79, data = ‘C’
host ACKs receipt of ‘C’, echoes back ‘C’
Seq=79, ACK=43, data = ‘C’
host ACKs receipt of echoed ‘C’
Seq=43, ACK=80

simple telnet scenario

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

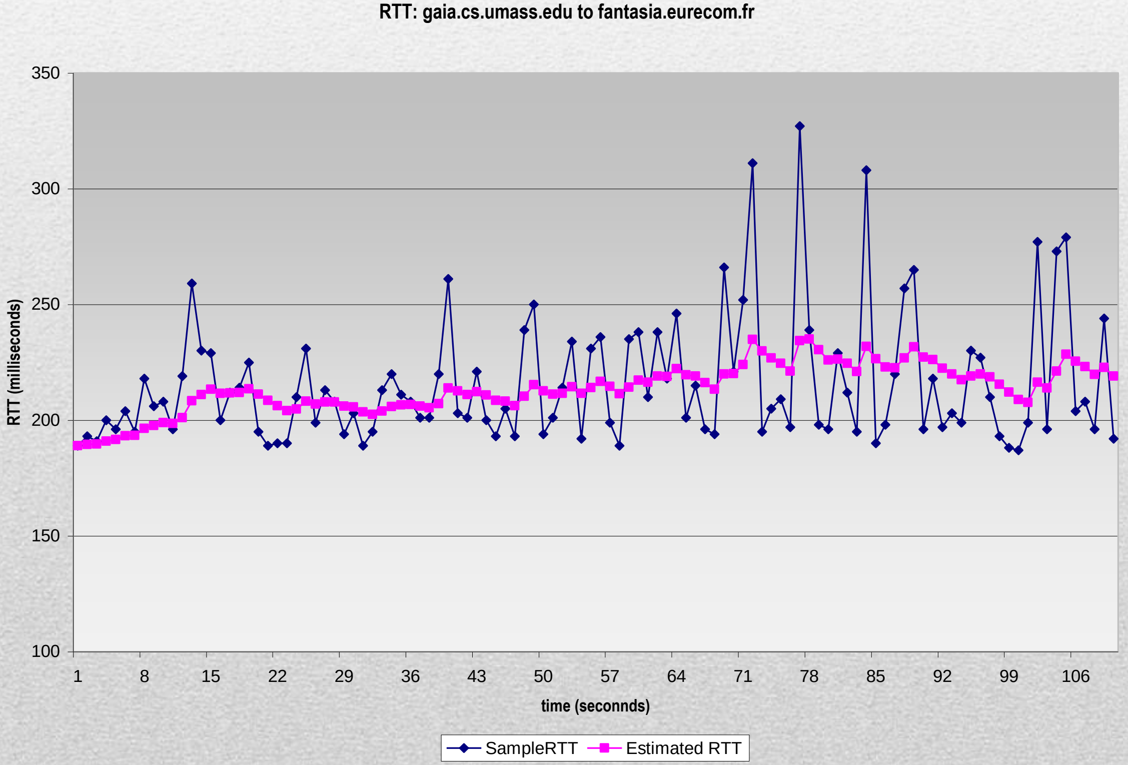
Q: How to estimate RTT?

- **`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
- Influence of past sample decreases exponentially fast
- Typical value: **α =** 0.125



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)
```

$$\texttt{TimeoutInterval = } \underset{\substack{\uparrow \\ \text{estimated RTT}}}{\texttt{EstimatedRTT}} \texttt{ + } \underset{\substack{\uparrow \\ \text{“safety margin”}}}{\texttt{4*DevRTT}}$$