COMP 3234B
Computer and Communication Networks

2nd semester 2023-2024

Transport Layer (II)

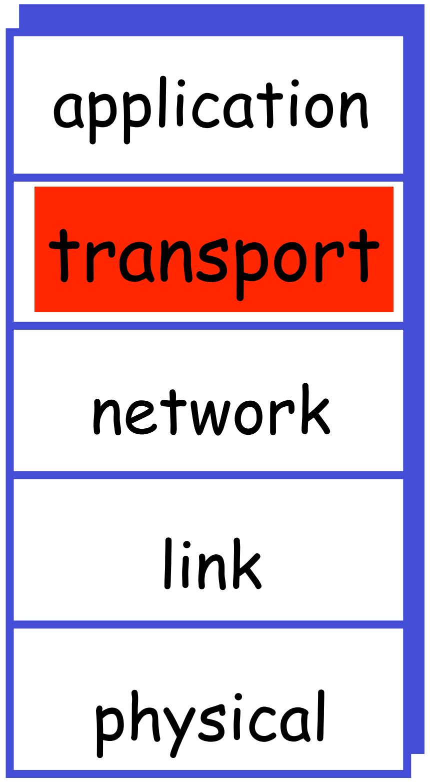
Prof. C. Wu

Department of Computer Science
The University of Hong Kong

Roadmap

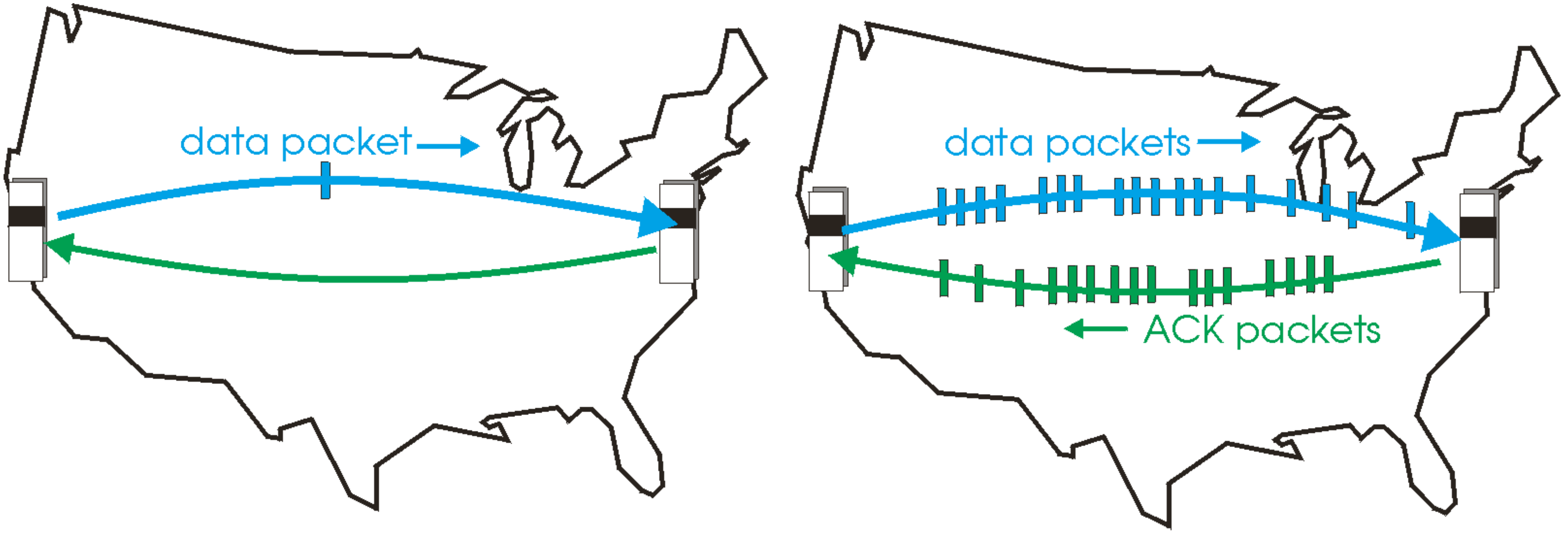
Transport layer

- Principles behind transport-layer services
  - multiplexing/demultiplexing (ILO1, 2)
  - reliable data transfer:(ILO 2, 3)
    - rdt 1.0
    - rdt 2.0, 2.1, 2.2
    - rdt 3.0
    - GBN
    - selective repeat
  - flow control (ILO 2, 3)
  - congestion control (ILO 2, 3)
- Transport protocols in the Internet (ILO 2, 3)
  - TCP
  - UDP



Pipelining

- Sender can send multiple packets without waiting for acknowledgements



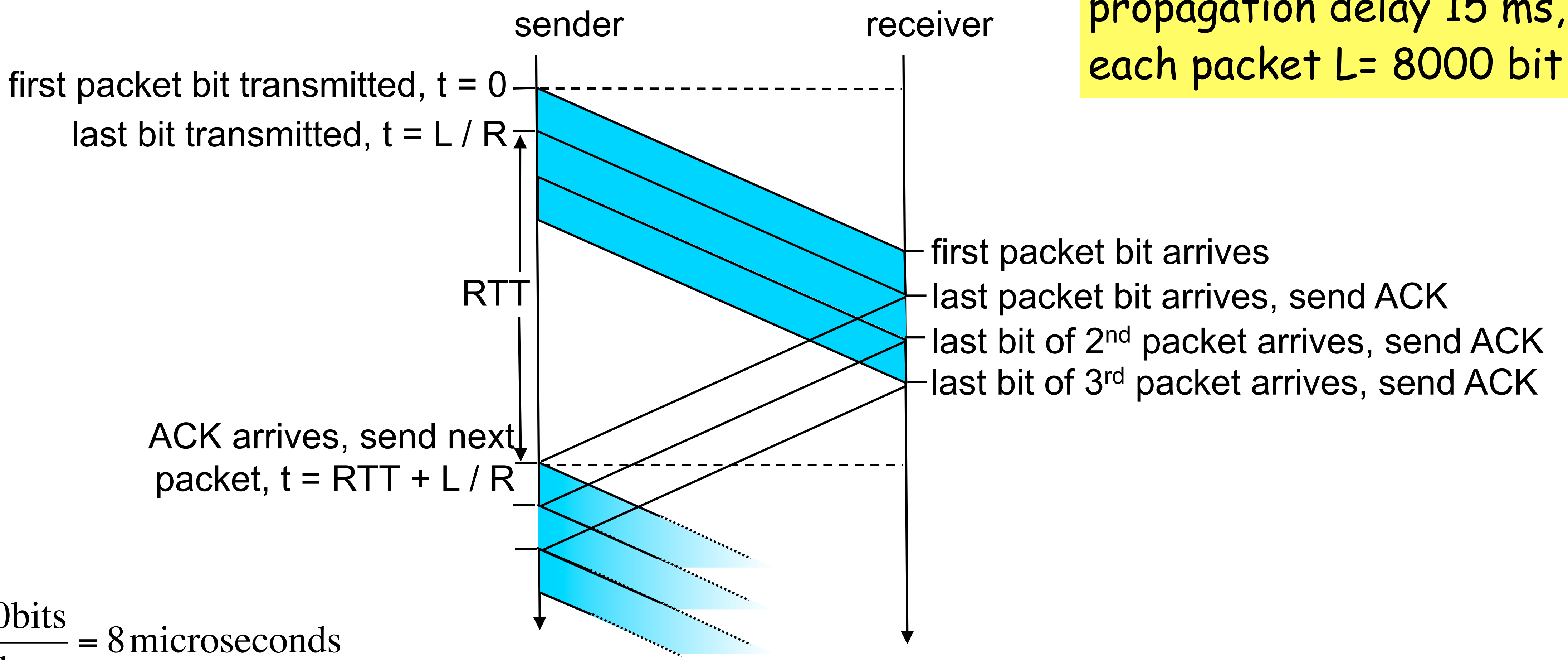
(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

Pipelining: performance

increased utilization

link rate R=1 Gbps,
propagation delay 15 ms,
each packet L= 8000 bit


sender
receiver
first packet bit transmitted, t = 0
last bit transmitted, t = L / R
RTT
ACK arrives, send next packet, t = RTT + L / R
first packet bit arrives
last packet bit arrives, send ACK
last bit of 2nd packet arrives, send ACK
last bit of 3rd packet arrives, send ACK

$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\,\text{microseconds}$$

$$RTT = 2 \times 15ms = 30ms$$

3-packet pipelining increases utilization by a factor of 3!

$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Pipelining protocols

- New requirements for rdt
  - range of sequence numbers must be increased
  - buffering more than one packet at sender and/or receiver

- Two generic forms of pipelined reliable data transfer protocols

Go-back-N

- Sender can have up to N unacked packets in pipeline
- Receiver only sends cumulative acks
- Sender has timer for oldest unacked packet
  - If timer expires, retransmit all unacked packets

Selective Repeat

- Sender can have up to N unacked packets in pipeline
- Receiver acks individual packets
- Sender maintains timer for each unacked packet
  - When timer expires, retransmit only unack packet

Go-Back-N (GBN): sender

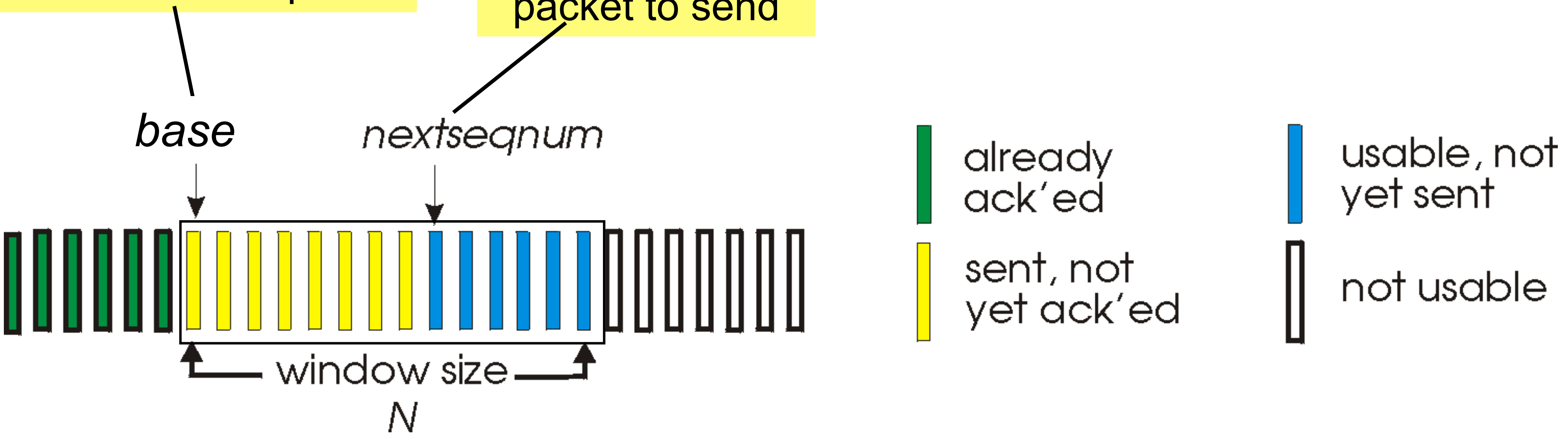
- Sender protocol
  - *k*-bit seq # carried in a field in packet header

    seq # range: 0 to $2^k - 1$
  - sender allows up to *N* consecutive unack'ed packets
  - sender has timer for the oldest unacked packet

    timeout(n): retransmit packet n and all packets in window with higher seq #

Seq # of the oldest unacked packet

Seq # of the next packet to send



Go-Back-N (GBN): sender (cont'd)

- Sender protocol
  - 4 intervals of seq #

    [*1, base - 1*]: packets already transmitted and acked

    [*base, nextseqnum-1*]: packets sent but not yet acked

    [*nextseqnum, base+N-1*]: packets that can be sent immediately (when data from upper layer arrive)

    [*base+N, --*]: cannot be used until an unacked packet has been acked

  - Window of size *N* slides forward over the seq # space

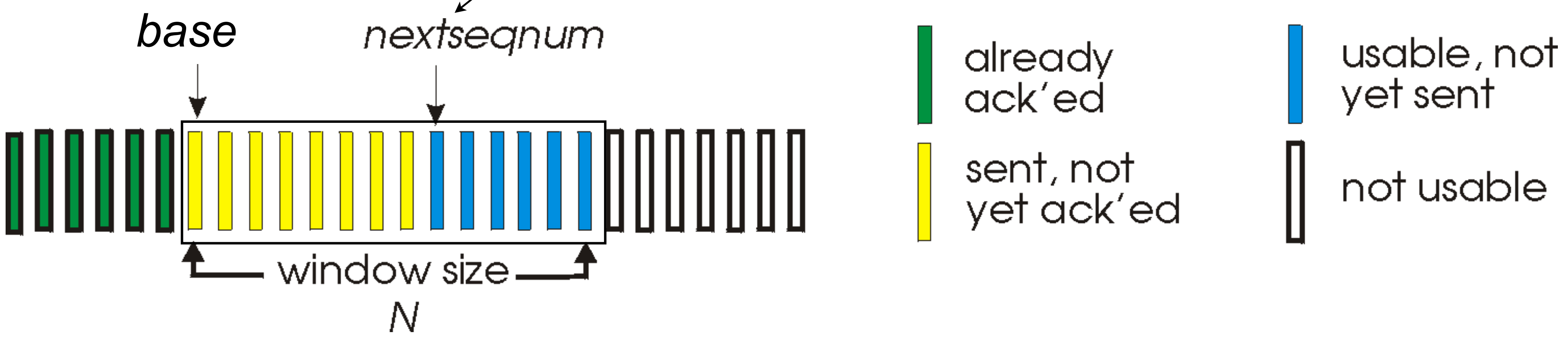
    GBN: sliding-window protocol

Q: why do we limit the number of unacked packets to *N*?

A: flow control, congestion control

Seq # of the oldest unacked packet

Seq # of the next packet to send



Go-Back-N (GBN): receiver

- Receiver protocol
  - sends ACK for correctly-received packet with highest in-order seq #

    i.e., cumulative ACK

    ACK(n): ACKs all packets up to, including seq # n
  - in any other case, discards packet and resends ACK for the most recently received packet with highest in-order seq #

    e.g., corrupted packet, out-of-order packet

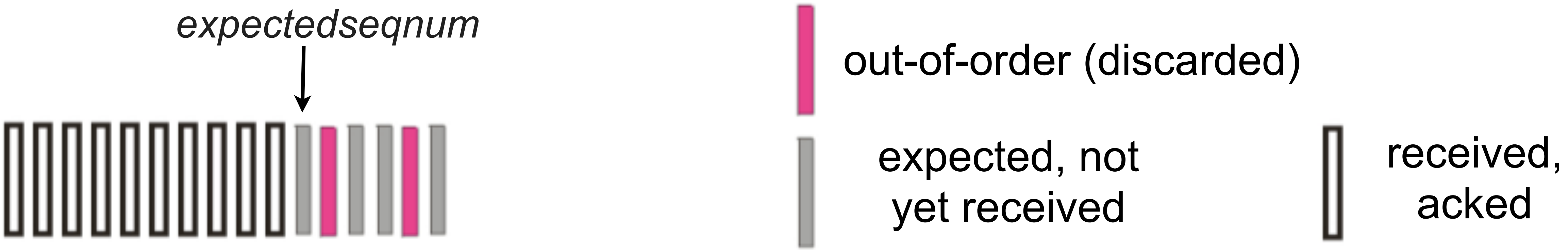
    => may generate duplicate ACKs

    Why discarding out-of-order packets? (e.g., if packet *n+1* is received while packet *n* is expected, discard packet *n+1*)

    reason: if packet *n* is lost, packet *n+1* will anyway be retransmitted

    advantage: no receiver buffering! it need only remember *expectedseqnum*

    disadvantage: more retransmissions needed if subsequent retransmission of packet *n+1* is lost or corrupted



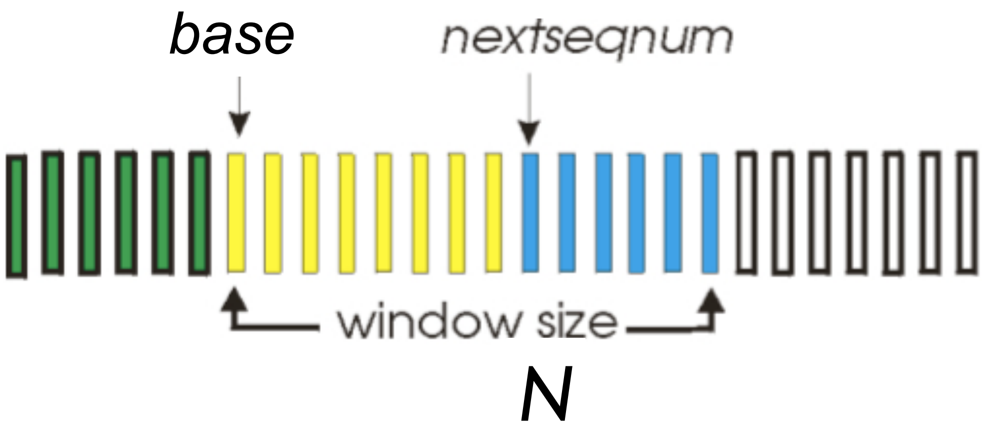
expectedseqnum

out-of-order (discarded)

expected, not yet received

received, acked

Go-Back-N (GBN): sender FSM



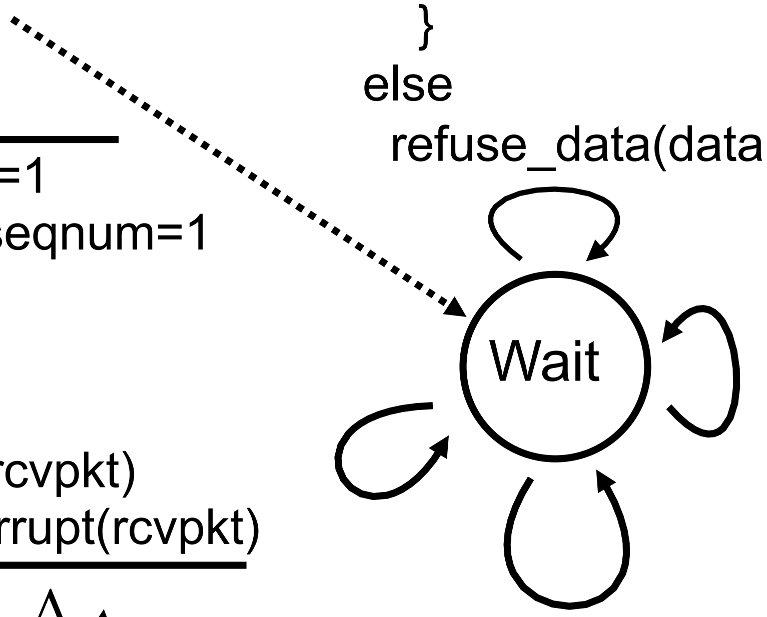
Sender FSM

invocation from above

```
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)
```

upper layer has to try again later

```
Λ
_____________
base=1
nextseqnum=1
```



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

a timeout event

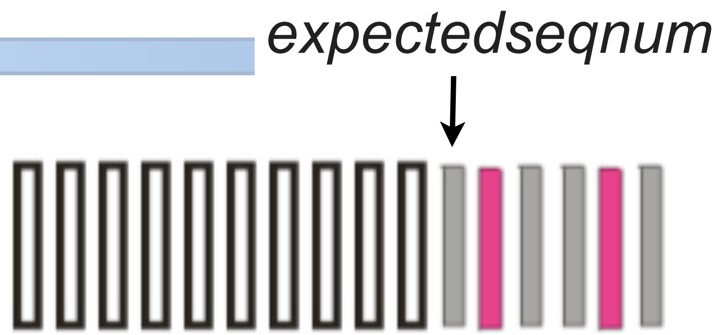
```
rdt_rcv(rcvpkt)
  && corrupt(rcvpkt)
_____________________
          Λ
```

receipt of an ACK

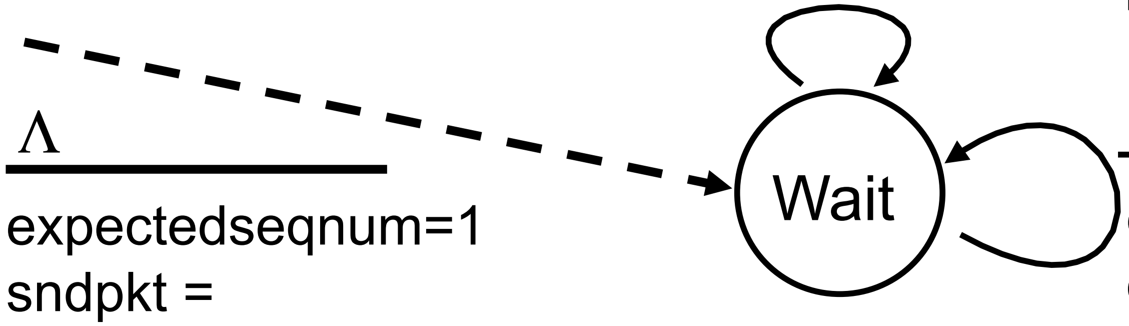
```
rdt_rcv(rcvpkt) &&
  notcorrupt(rcvpkt)
_____________________
base = getacknum(rcvpkt)+1
If (base == nextseqnum)
    stop_timer
  else
    start_timer
```

start timer when there are additional transmitted but unacked packets whose timer has not been started

Go-Back-N (GBN): receiver FSM



Receiver FSM



any other cases

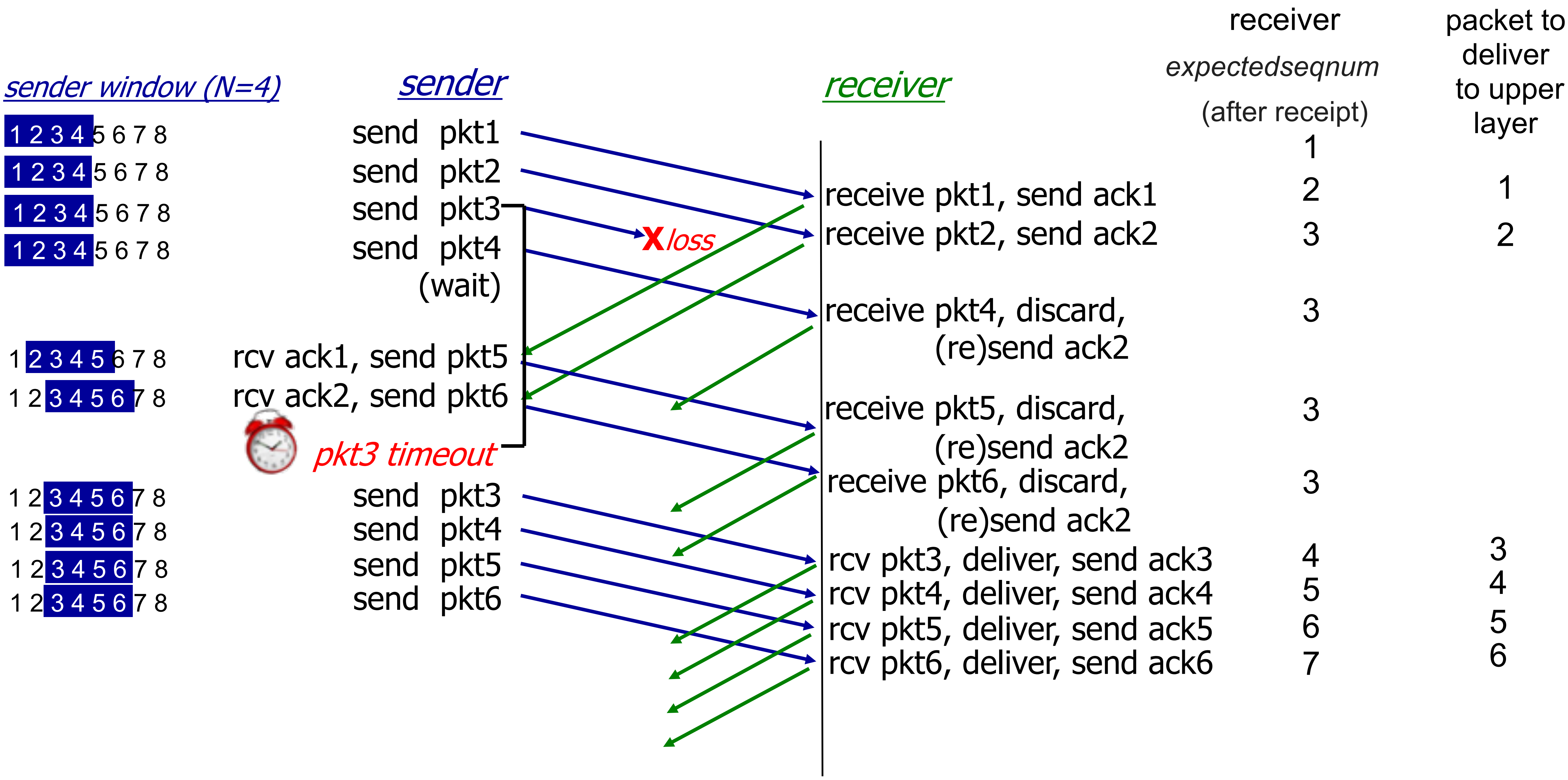
```
default
udt_send(sndpkt)
```

correctly receiving packet with in-order seq #

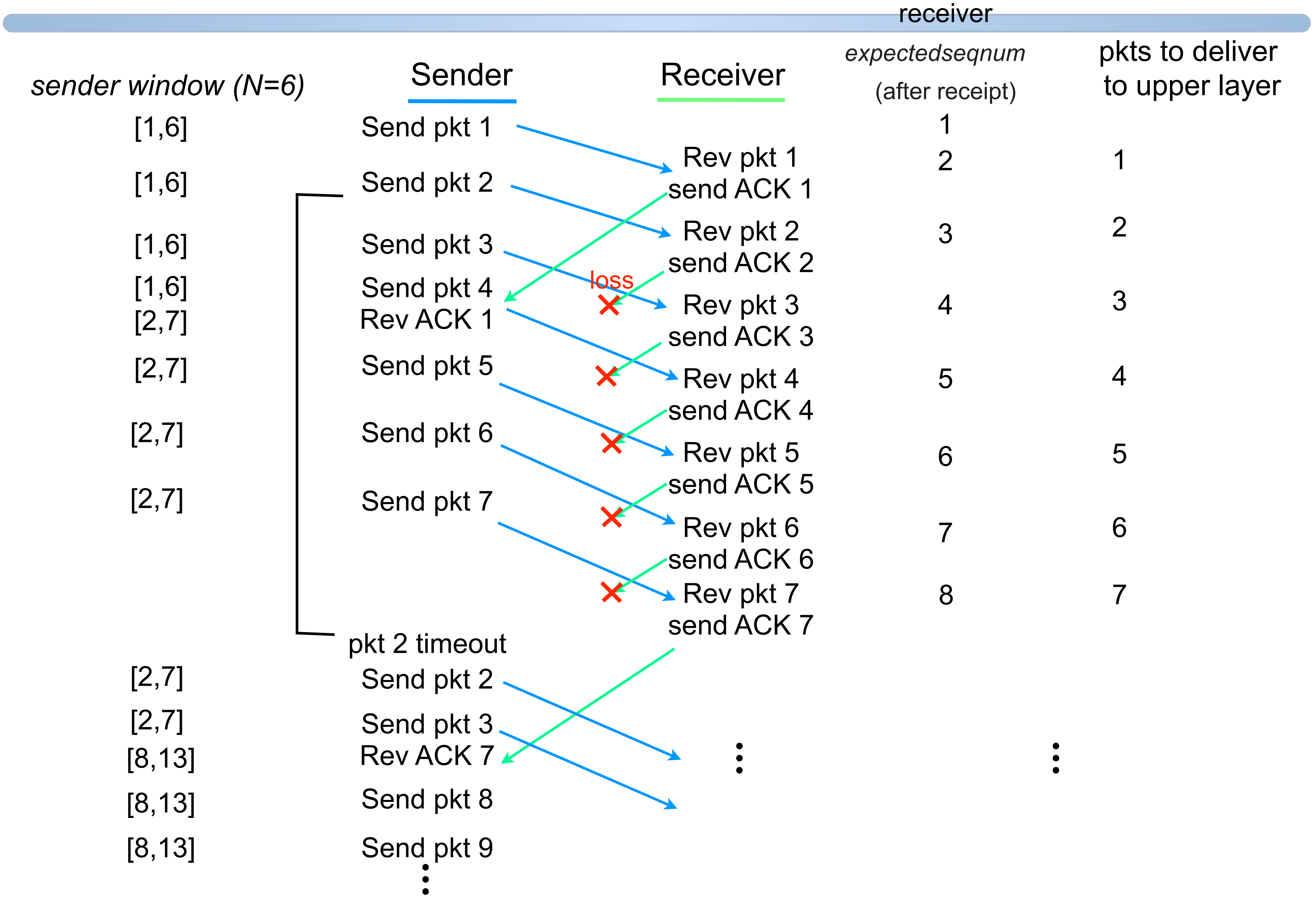
```
rdt_rcv(rcvpkt)
  && notcurrupt(rcvpkt)
  && hasseqnum(rcvpkt,expectedseqnum)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(expectedseqnum,ACK,chksum)
udt_send(sndpkt)
expectedseqnum++
```

```
Λ
expectedseqnum=1
sndpkt =
  make_pkt(0,ACK,chksum)
```

GBN in action (1)



GBN in action (2)



sender window (N=6)	Sender	Receiver	receiver *expectedseqnum* (after receipt)	pkts to deliver to upper layer
[1,6]	Send pkt 1		1	
		Rev pkt 1 send ACK 1	2	1
[1,6]	Send pkt 2			
		Rev pkt 2 send ACK 2	3	2
[1,6]	Send pkt 3			
[1,6]	Send pkt 4	loss		
		Rev pkt 3 send ACK 3	4	3
[2,7]	Rev ACK 1			
[2,7]	Send pkt 5			
		Rev pkt 4 send ACK 4	5	4
[2,7]	Send pkt 6			
		Rev pkt 5 send ACK 5	6	5
[2,7]	Send pkt 7			
		Rev pkt 6 send ACK 6	7	6
		Rev pkt 7 send ACK 7	8	7
	pkt 2 timeout			
[2,7]	Send pkt 2			
[2,7]	Send pkt 3			
[8,13]	Rev ACK 7	⋮		⋮
[8,13]	Send pkt 8			
[8,13]	Send pkt 9			
	⋮			

Go-Back-N (GBN): a major problem

- A major problem
  - many packets in pipeline when

    window size large and bandwidth-delay product large
  - single packet error causes retransmission of many packets

    not necessary

Selective Repeat

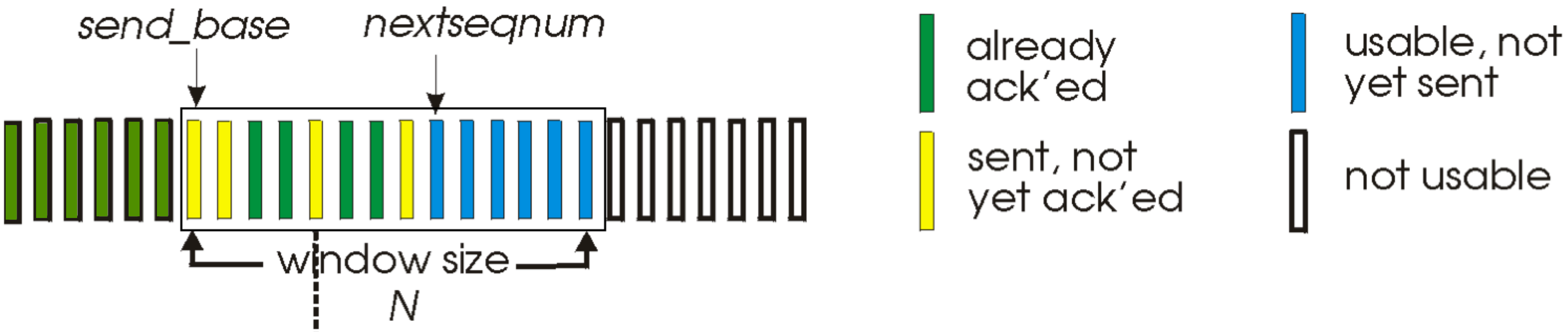
- Receiver individually acknowledges all correctly received packets
  - buffers packets as needed, for eventual in-order delivery to upper layer
- Sender only resends packets for which ACK not (correctly) received
  - sets timer for each unACKed packet
  - sender window

    N consecutive seq #'s

    limits seq #s of sent, unACKed pkts

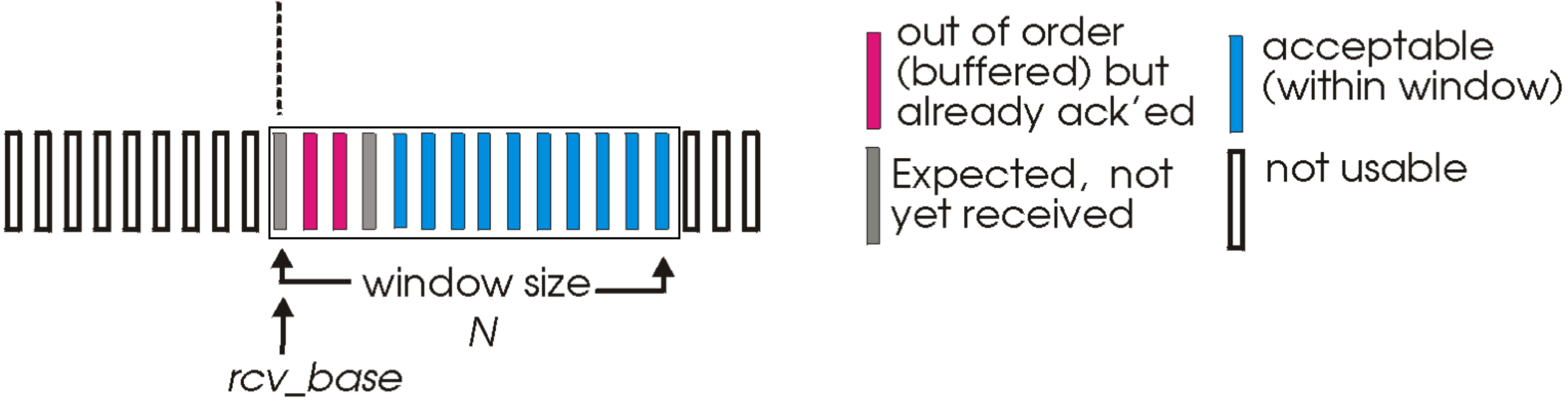
Selective Repeat: sender/receiver windows

Sender window



(a) sender view of sequence numbers

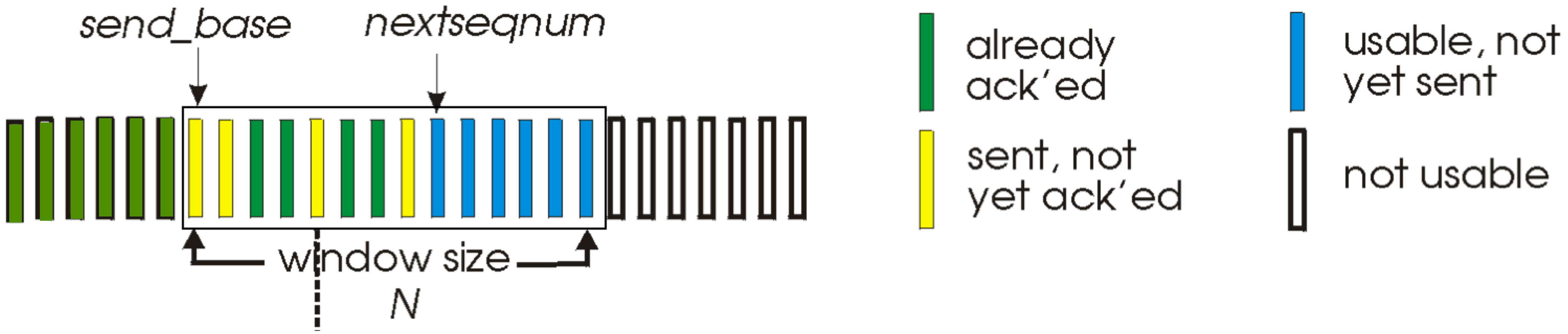
Receiver window



window size
N
rcv_base
out of order (buffered) but already ack'ed
Expected, not yet received
acceptable (within window)
not usable

(b) receiver view of sequence numbers

Selective Repeat: sender protocol



sender

data received from above:

- if next available seq # in window, send pkt and start timer on the pkt; otherwise, refuse data

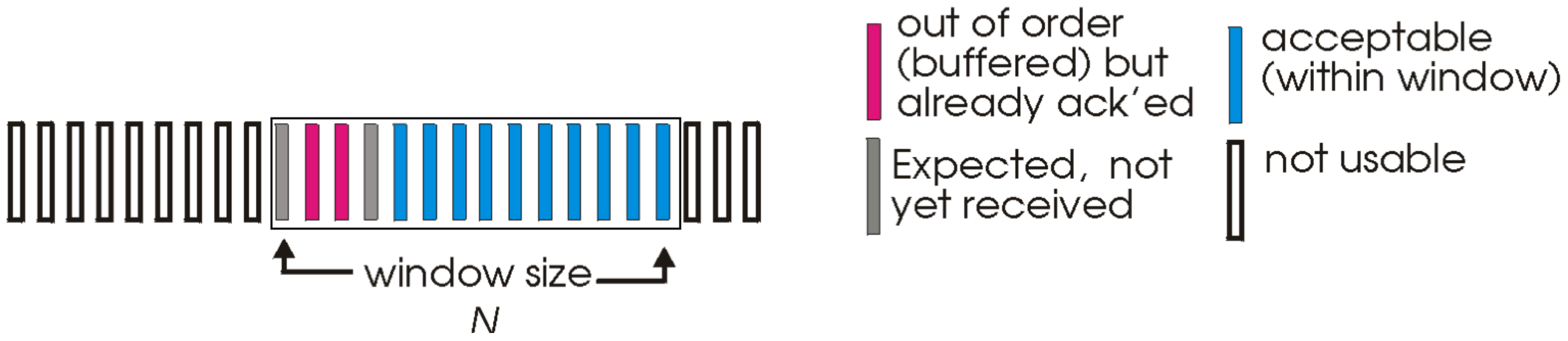
timeout(n):

- resend pkt n, restart timer on n

ACK(n) in [send_base,send_base+N-1] correctly received:

- mark pkt n as received and stop its timer
- if n is smallest unACKed pkt, advance send_base to next unACKed seq #

Selective Repeat: receiver protocol



window size
N
out of order (buffered) but already ack'ed
acceptable (within window)
Expected, not yet received
not usable

receiver

pkt n in [rcv_base, rcv_base+N-1]

correctly received:

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

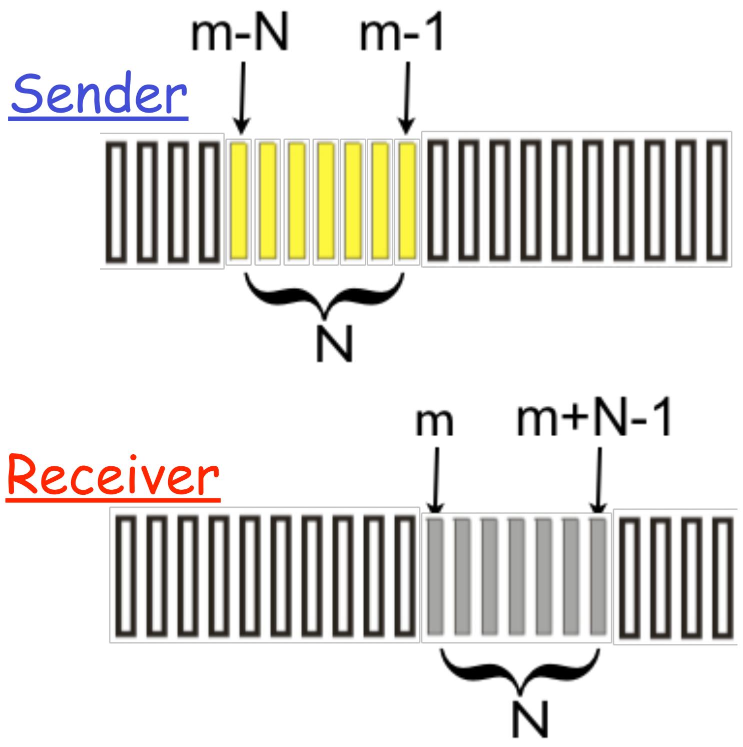
pkt n in [rcv_base-N,rcv_base-1]

- Send ACK(n)

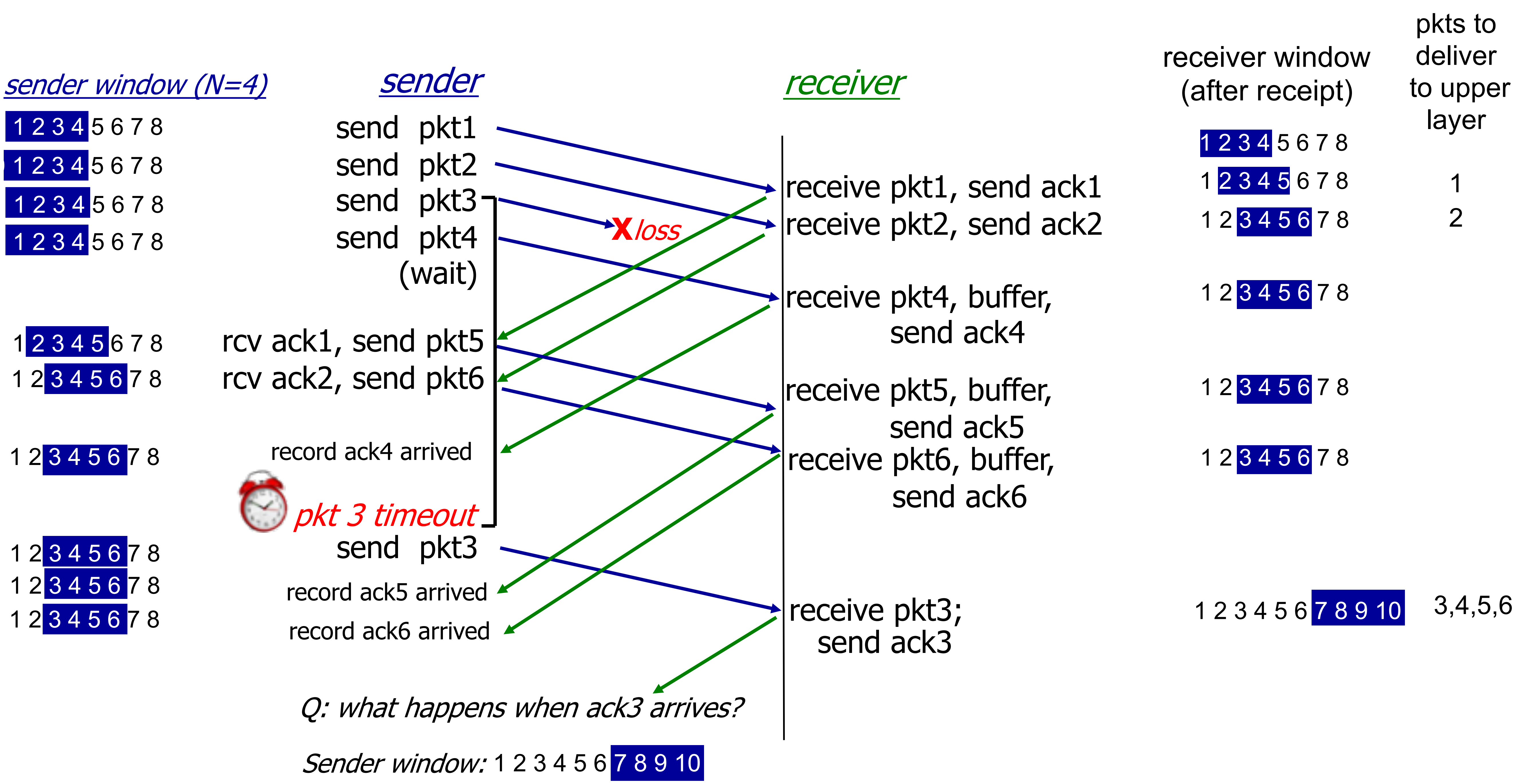
Why?

otherwise:

- ignore



Selective Repeat in action (1)



sender window (N=4)

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

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

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

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

sender

send pkt1
send pkt2
send pkt3
send pkt4
(wait)

rcv ack1, send pkt5
rcv ack2, send pkt6

record ack4 arrived

pkt 3 timeout
send pkt3
record ack5 arrived
record ack6 arrived

X *loss*

receiver

receive pkt1, send ack1
receive pkt2, send ack2

receive pkt4, buffer,
send ack4

receive pkt5, buffer,
send ack5
receive pkt6, buffer,
send ack6

receive pkt3;
send ack3

receiver window
(after receipt)

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

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

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

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

1 2 3 4 5 6 **7 8 9 10**

pkts to
deliver
to upper
layer

1
2

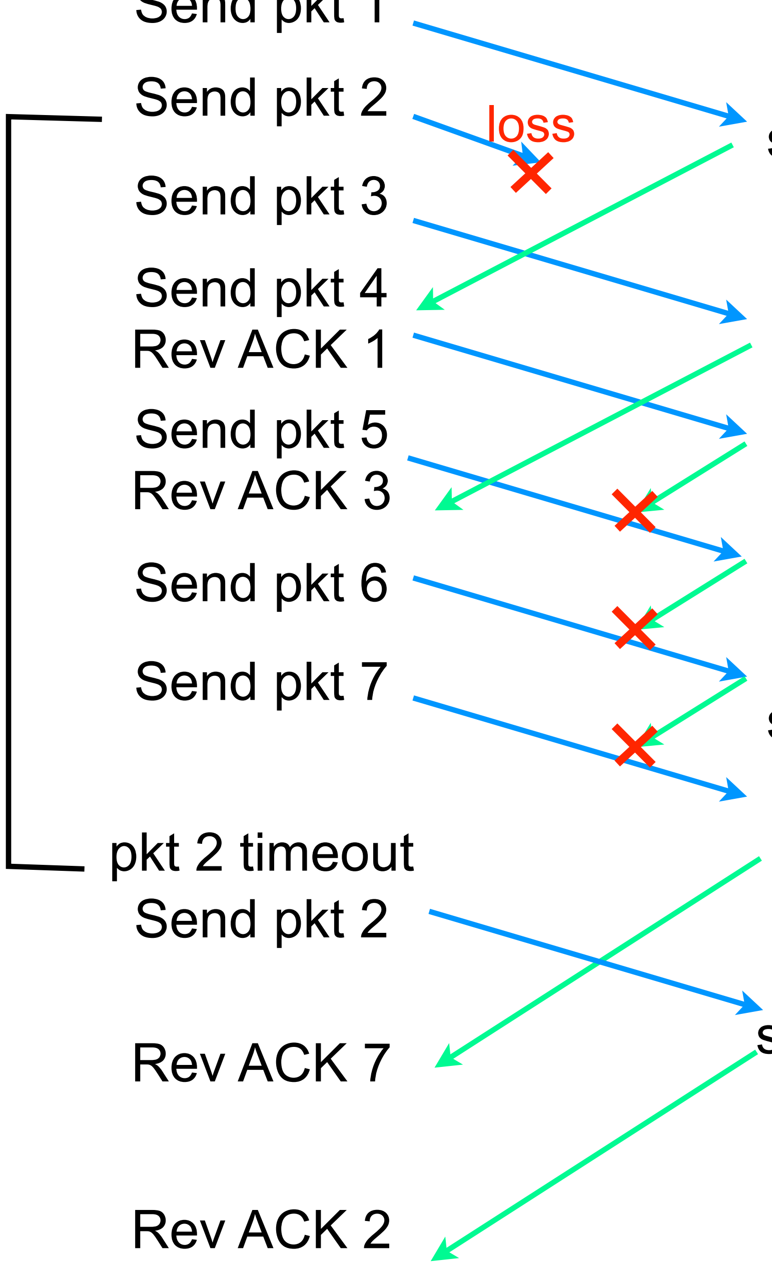
3,4,5,6

Q: what happens when ack3 arrives?

Sender window: 1 2 3 4 5 6 **7 8 9 10**

Selective Repeat in action (2)

N=6



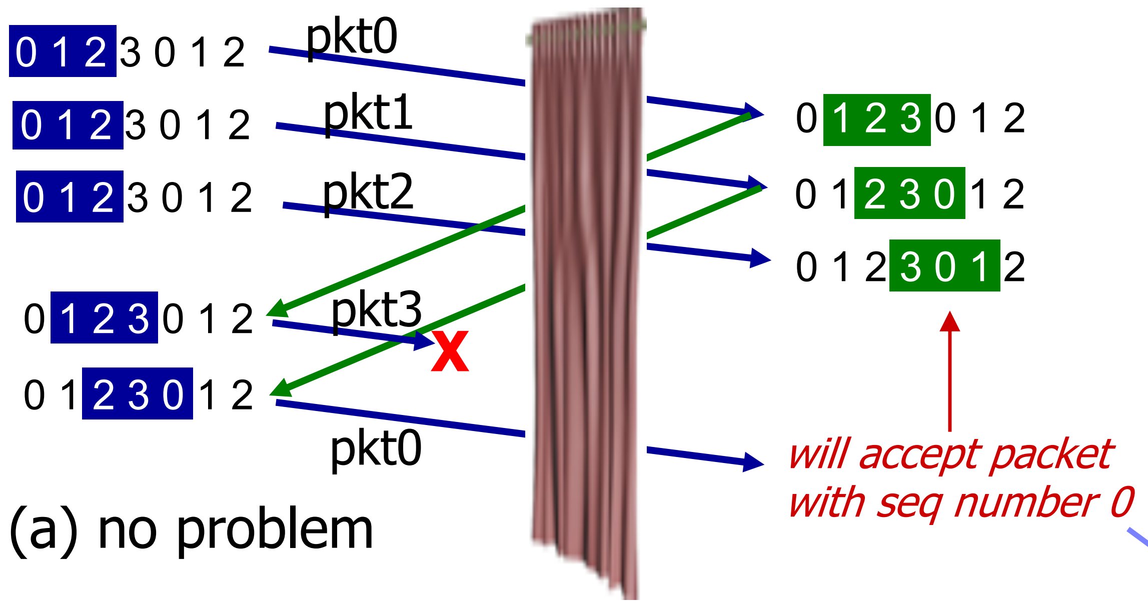
ACKs in buffer	sender window	Sender
	[1,6]	Send pkt 1
	[1,6]	Send pkt 2
	[1,6]	Send pkt 3
	[1,6] [2,7]	Send pkt 4 Rev ACK 1
3	[2,7] [2,7]	Send pkt 5 Rev ACK 3
	[2,7]	Send pkt 6
	[2,7]	Send pkt 7
		pkt 2 timeout
	[2,7]	Send pkt 2
3, 7	[2,7]	Rev ACK 7
7	[4,9]	Rev ACK 2 Send pkt 8
		⋮

Receiver	receiver window (after receipt)	pkts in buffer	pkts to deliver to upper layer
	[1,6]		
Rev pkt 1 send ACK 1	[2,7]		1
Rev pkt 3 send ACK 3	[2,7]	3	
Rev pkt 4 send ACK 4	[2,7]	3, 4	
Rev pkt 5 send ACK 5	[2,7]	3,4,5	
Rev pkt 6 send ACK 6	[2,7]	3,4,5,6	
Rev pkt 7 send ACK 7	[2,7]	3,4,5,6,7	
Rev pkt 2 send ACK 2	[8,13]		2,3,4,5,6,7
⋮			

Selective Repeat: window size vs. seq # size

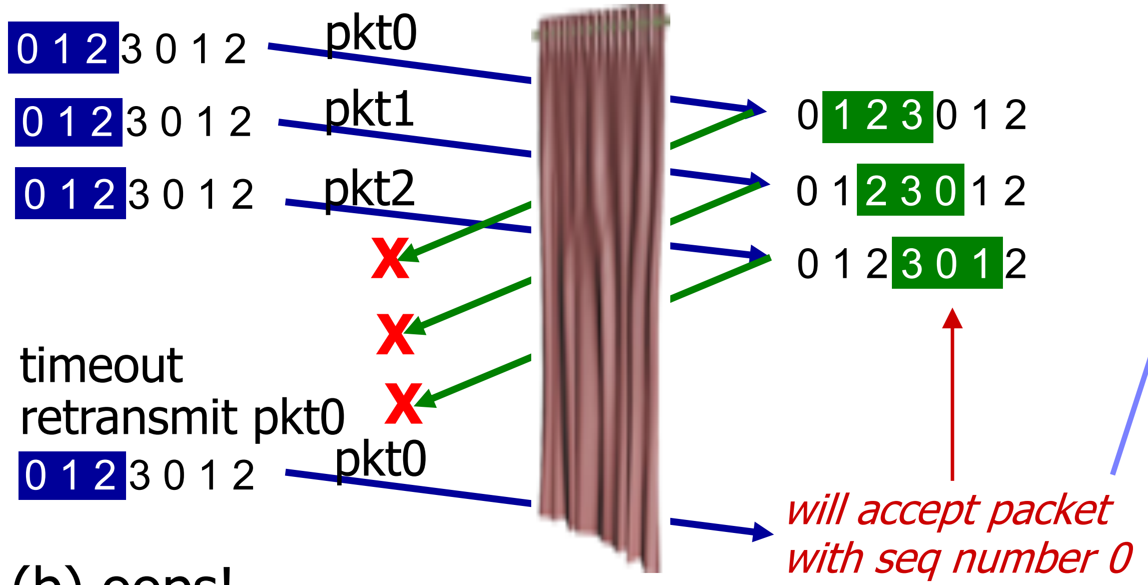
sender window (after receipt)

receiver window (after receipt)



(a) no problem

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



(b) oops!

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

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

Receiver sees no difference in two scenarios!

in (b), incorrectly passes duplicate data as new

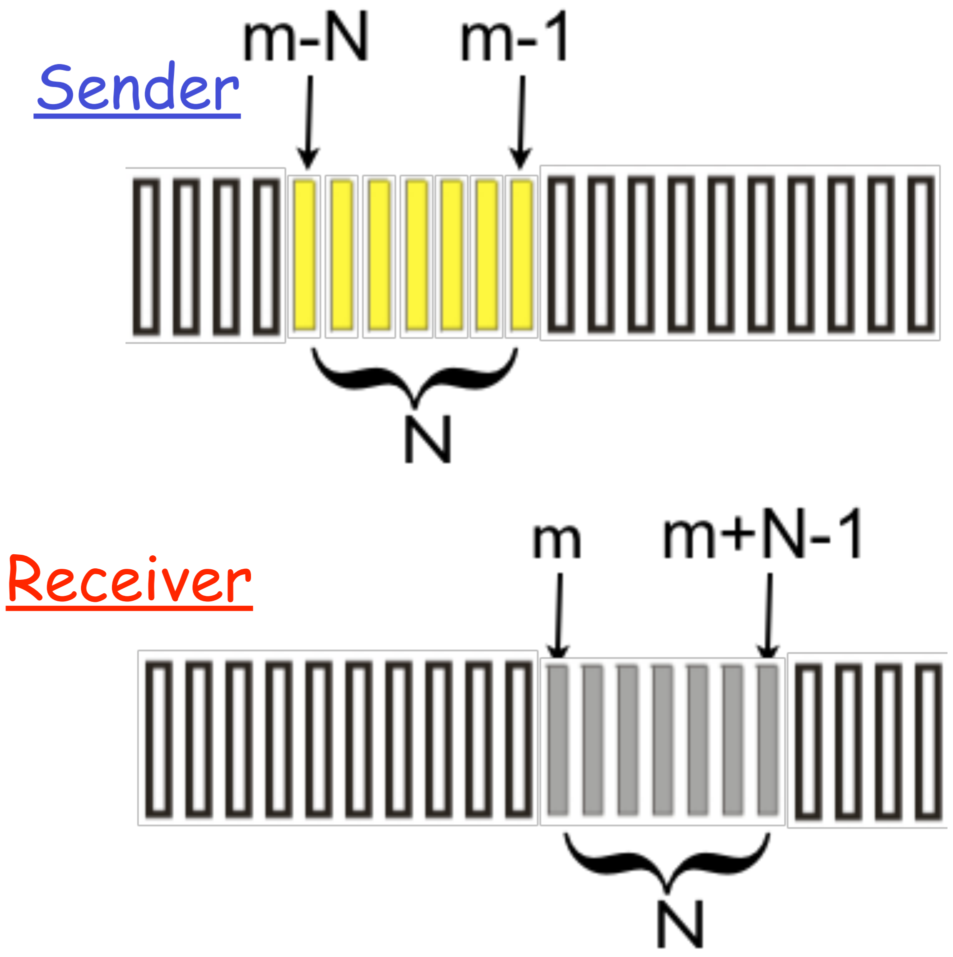
Selective Repeat: window size vs. seq # size (cont'd)

- Relationship between window size *N* and seq. # size *q* (2^k)
  - seq. # space must be large enough to fit the entire receiver window and the entire sender window
  - The extreme scenario

    receiver expects pkts [*m, m+N-1*]

    ACKs for pkt [*m-N,m-1*] are still propagating back

    sender window [*m-N,m-1*]



Therefore

Selective Repeat: $q \geq 2N$

Others

GBN: $q \geq N$+1

Stop-and-Wait: $q \geq 2$

- Required reading
  - *Computer Networking: A Top-Down Approach* (8th Edition) Ch 3.4.2, 3.4.3, 3.4.4
- Interactive animation of GBN:
  - https://media.pearsoncmg.com/ph/esm/ecs_kurose_compnetwork_8/cw/content/interactiveanimations/go-back-n-protocol/index.html
- Interactive animation of SR:
  - https://media.pearsoncmg.com/ph/esm/ecs_kurose_compnetwork_8/cw/content/interactiveanimations/selective-repeat-protocol/index.html
- Acknowledgement:
  - Some materials are extracted from the slides created by Prof. Jim F. Kurose and Prof. Keith W. Ross for the textbook.