AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY
श्रद्धावान् लभते ज्ञानम्

COMPUTER NETWORKS
3-0-0 3

TRANSPORT LAYER

Chapter 3: Transport Layer

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

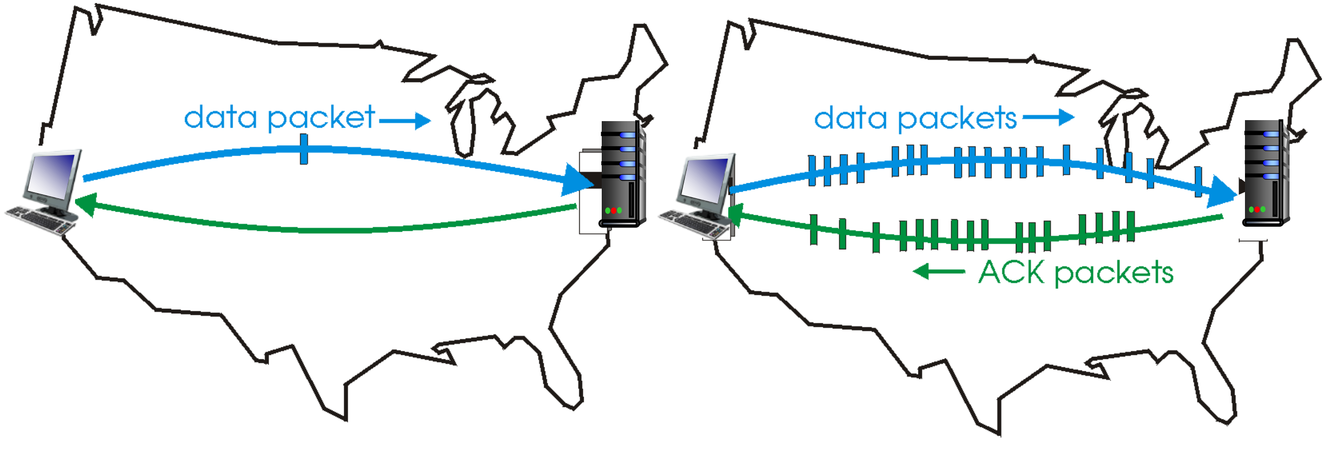
3.6 principles of congestion control

3.7 TCP congestion control

Pipelined protocols

pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver

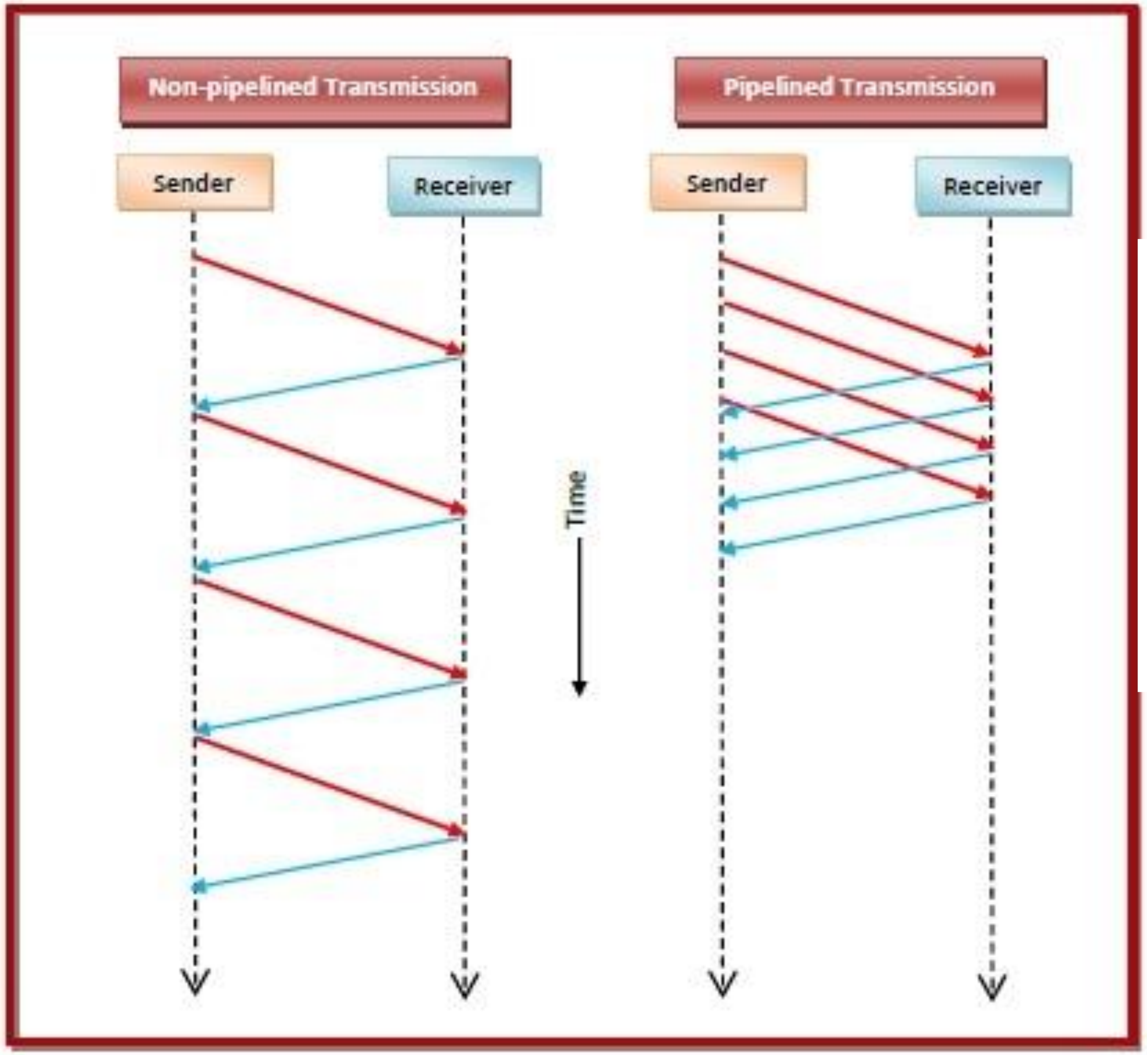


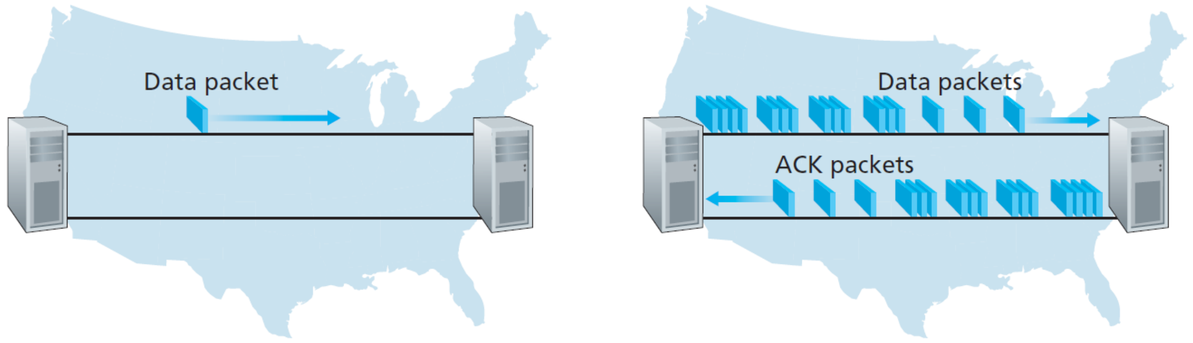
(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Sliding Window Protocols

- two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Non pipelined and pipelined

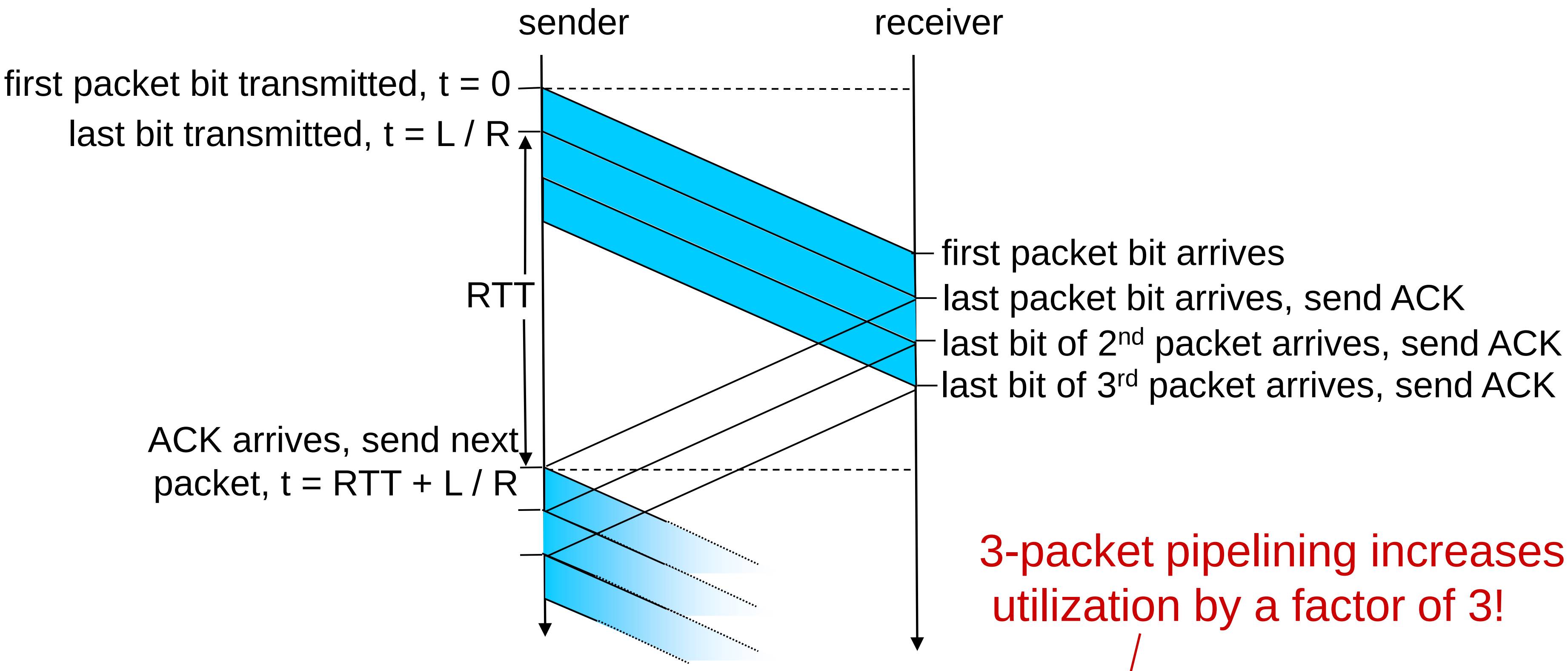




a. A stop-and-wait protocol in operation

b. A pipelined protocol in operation

Pipelining: increased utilization


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

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

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Pipelined protocols: **Sliding Window Protocols**

Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

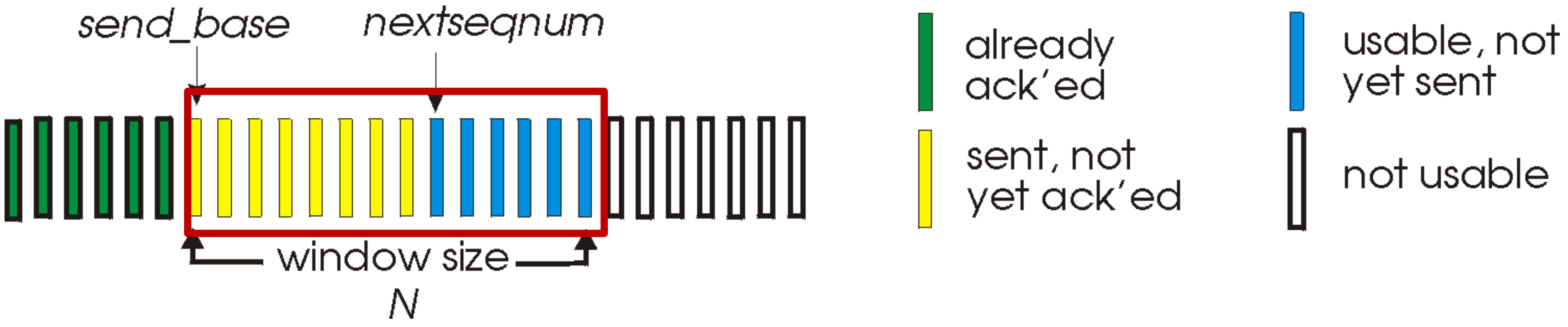
Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet

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

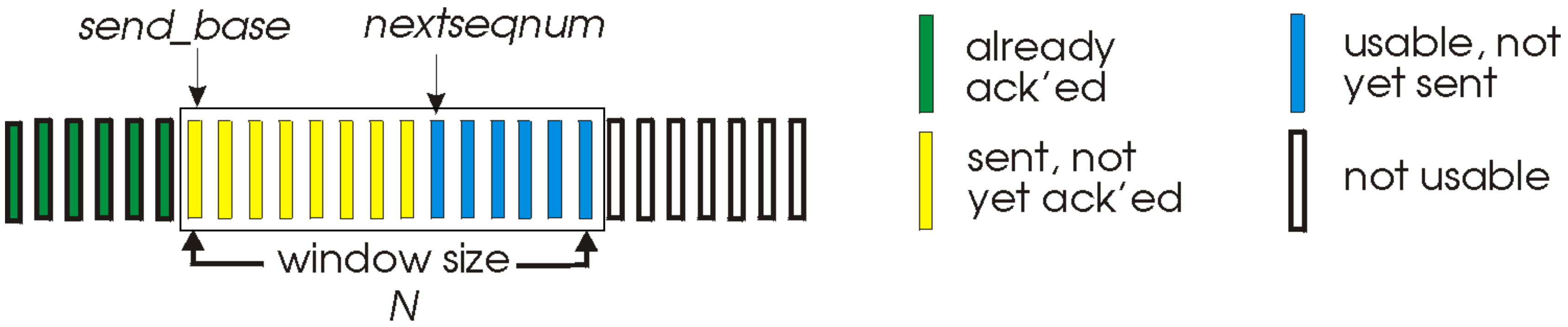
Go-Back-N: sender

- **Base** to be the sequence number of the oldest unacknowledged packet
- **nextseqnum** to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent), then four intervals in the range of sequence numbers can be identified.
- Sequence numbers in the **interval [0, base-1]** correspond to packets that have already been transmitted and acknowledged.
- The **interval [base, nextseqnum-1]** corresponds to packets that have been sent but not yet acknowledged. Sequence numbers in the **interval [nextseqnum, base+N-1]** can be used for packets that can be sent immediately, should data arrive from the upper layer.
- In the end, sequence numbers **greater than or equal to base+N** cannot be used until an unacknowledged packet currently in the pipeline (particularly, the packet with sequence number base) has been acknowledge



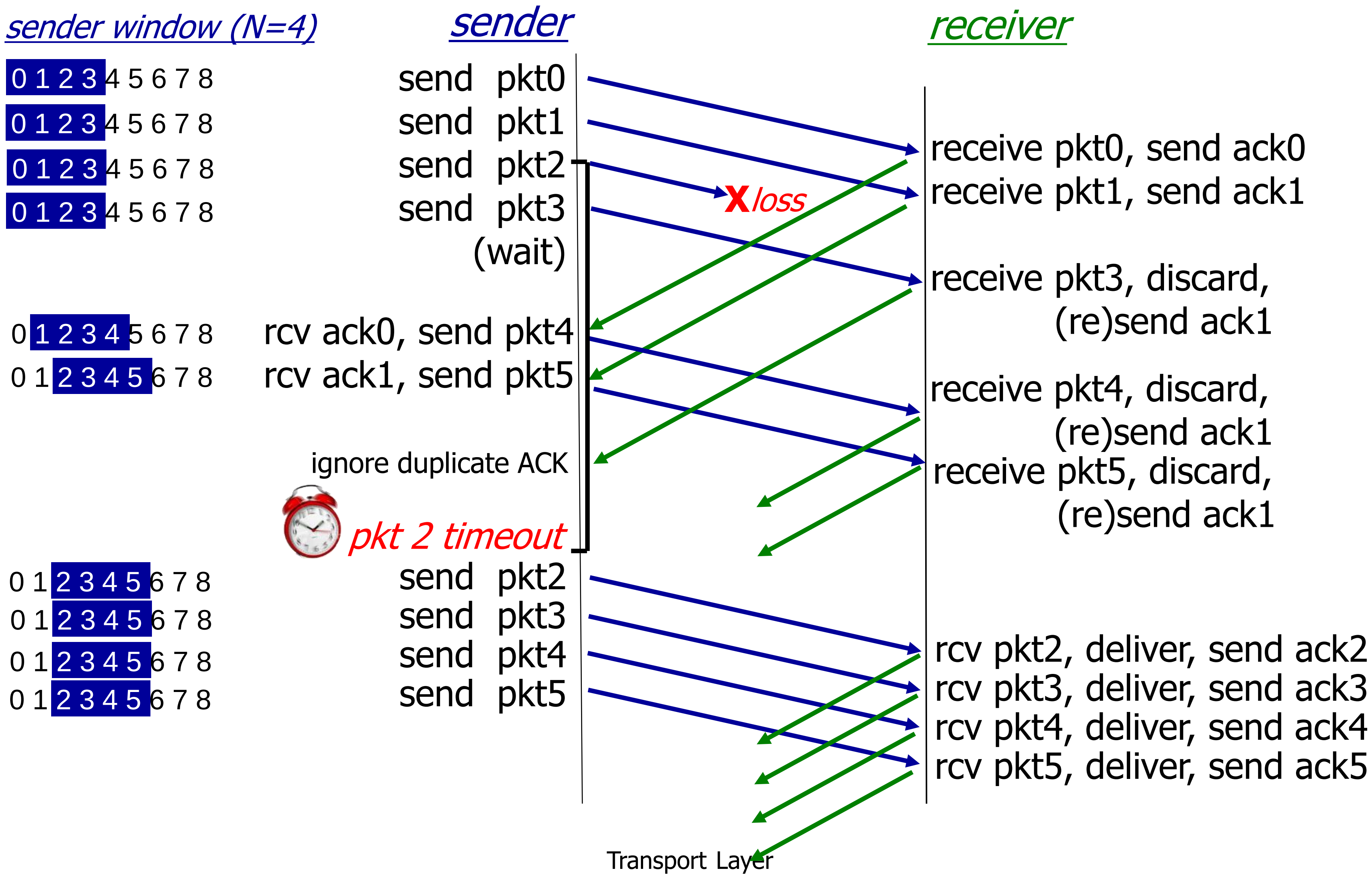
Go-Back-N: sender

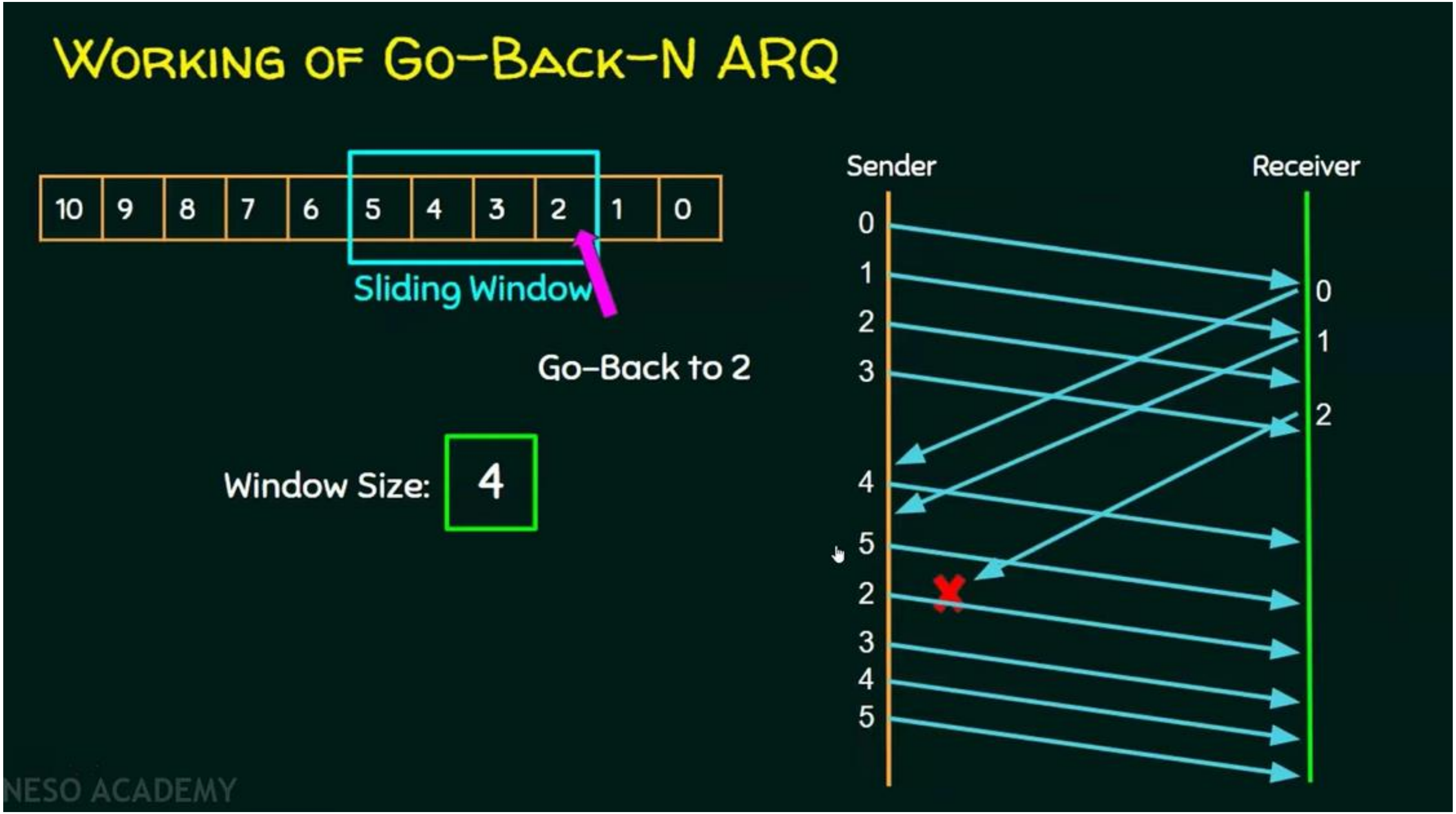
- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed



- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

GBN in action





Working of Go-Back-N ARQ

10	9	8	7	6	5	4	3	2	1	0

Sliding Window

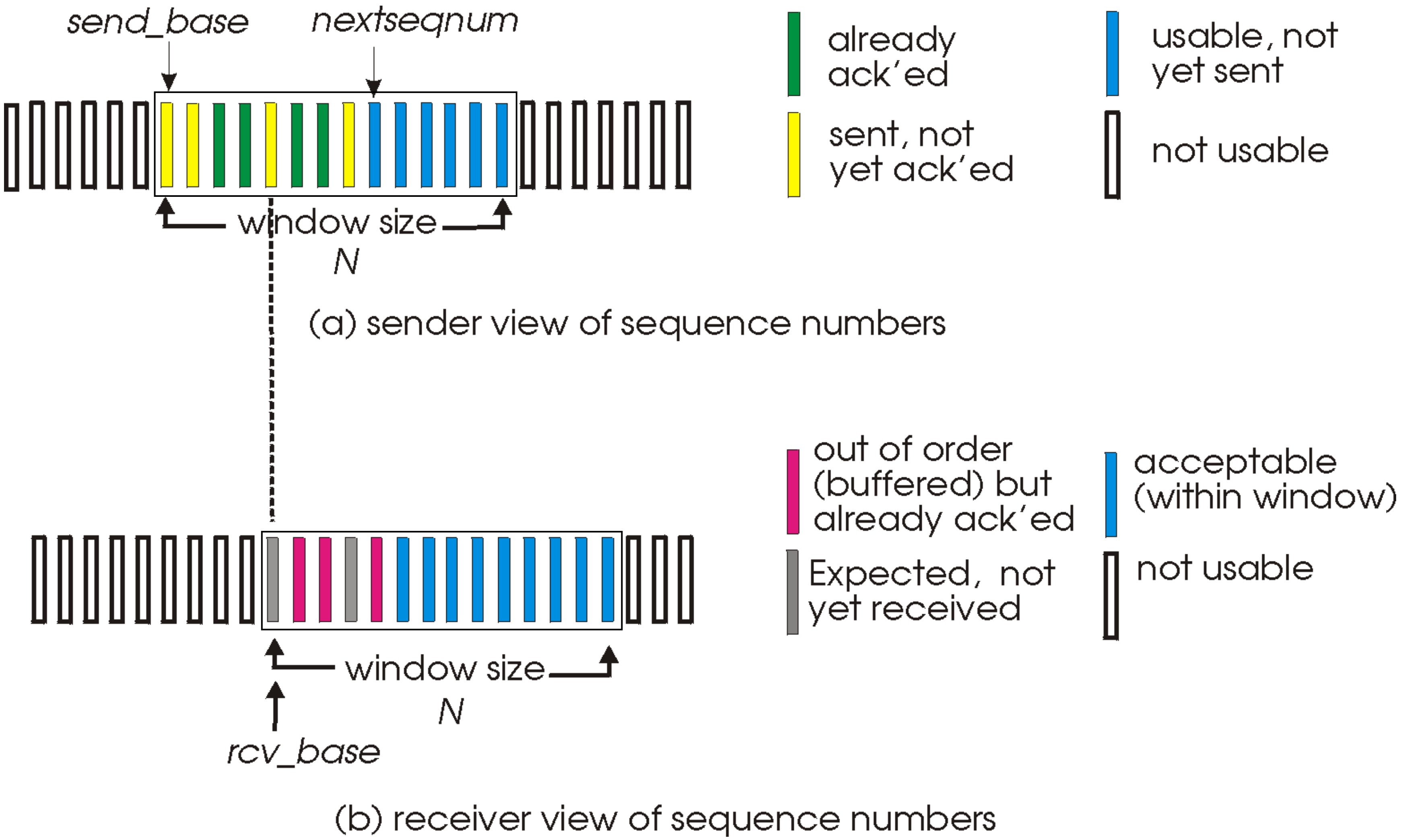
Go-Back to 2

Window Size: 4

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

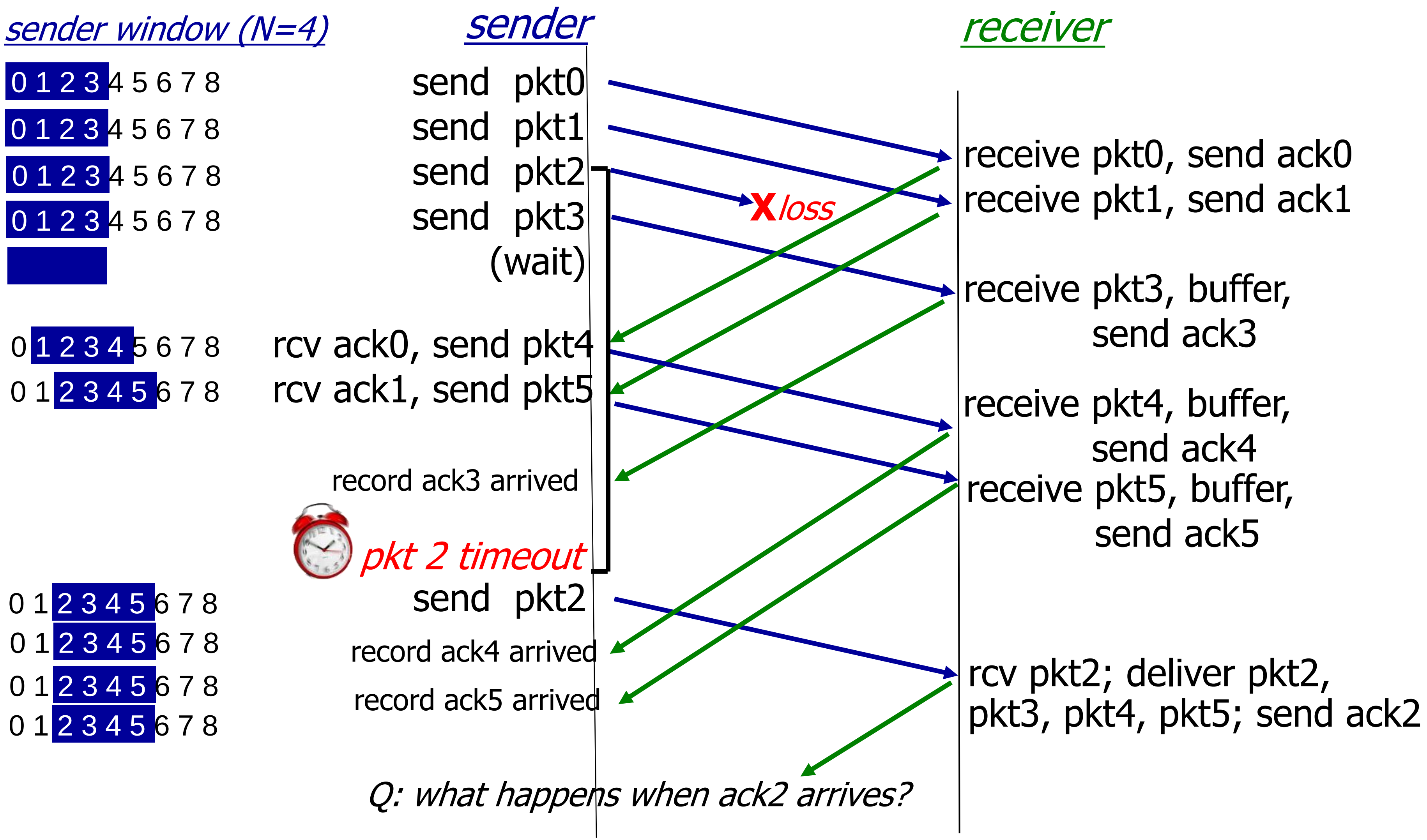
Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

(b) receiver view of sequence numbers

Selective repeat in action



sender window (N=4)

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

sender

send pkt0

send pkt1

send pkt2

send pkt3

(wait)

rcv ack0, send pkt4

rcv ack1, send pkt5

record ack3 arrived

pkt 2 timeout

send pkt2

record ack4 arrived

record ack5 arrived

receiver

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, buffer, send ack3

receive pkt4, buffer, send ack4

receive pkt5, buffer, send ack5

rcv pkt2; deliver pkt2, pkt3, pkt4, pkt5; send ack2

Q: what happens when ack2 arrives?

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

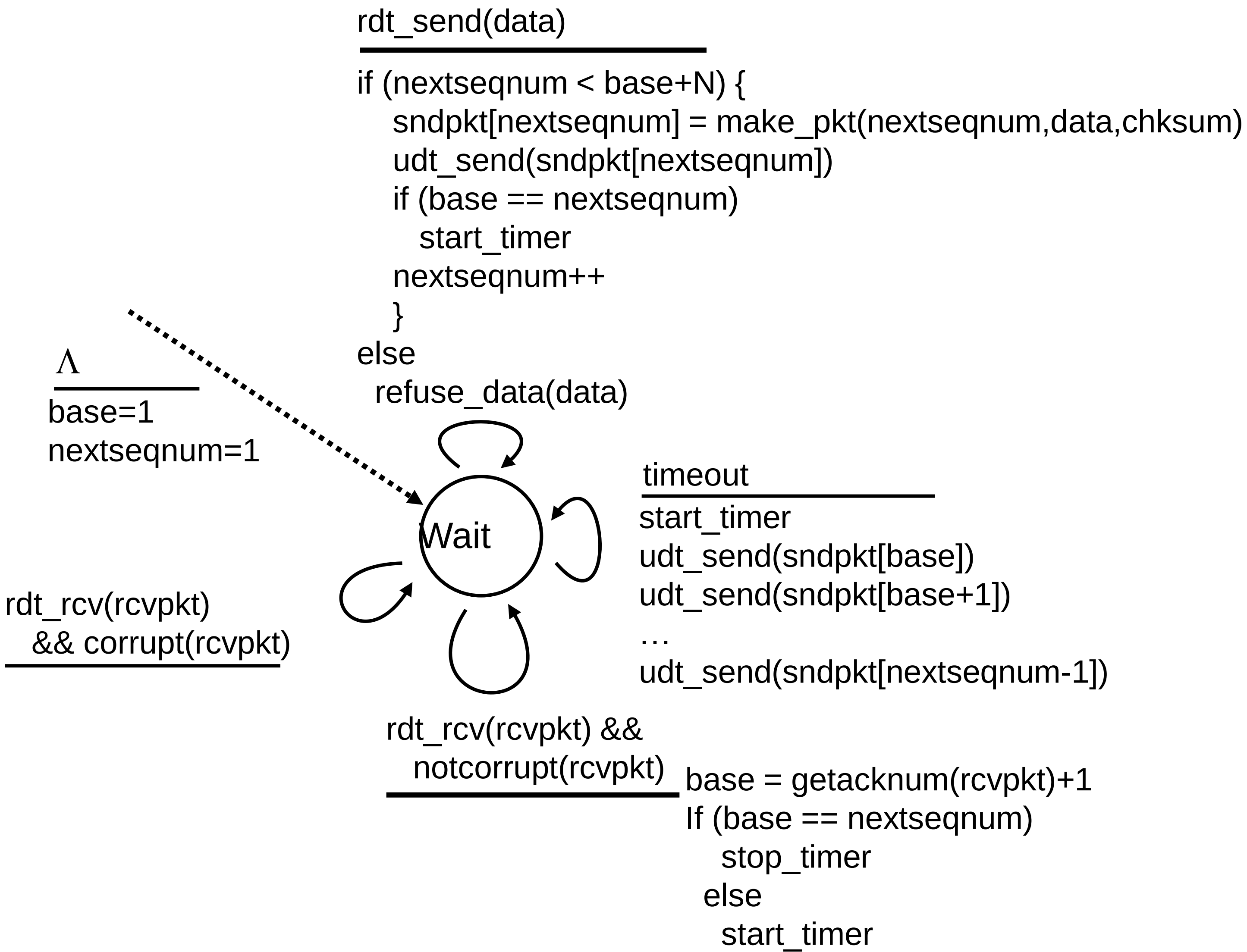
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

GBN: sender extended 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)
```

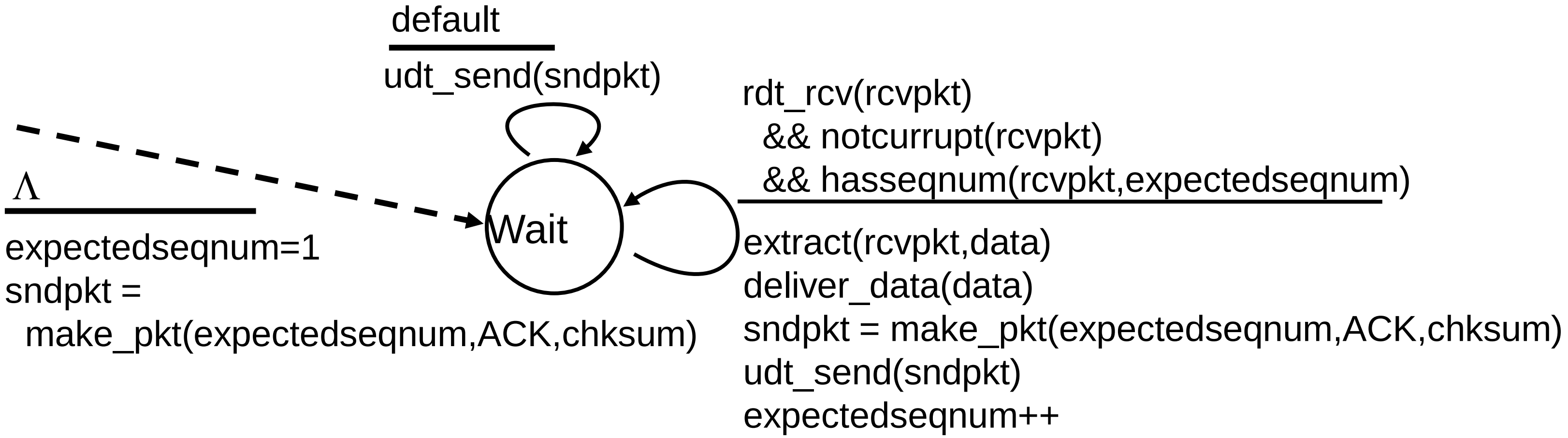
```
Λ
__________
base=1
nextseqnum=1
```

```
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 extended FSM



ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

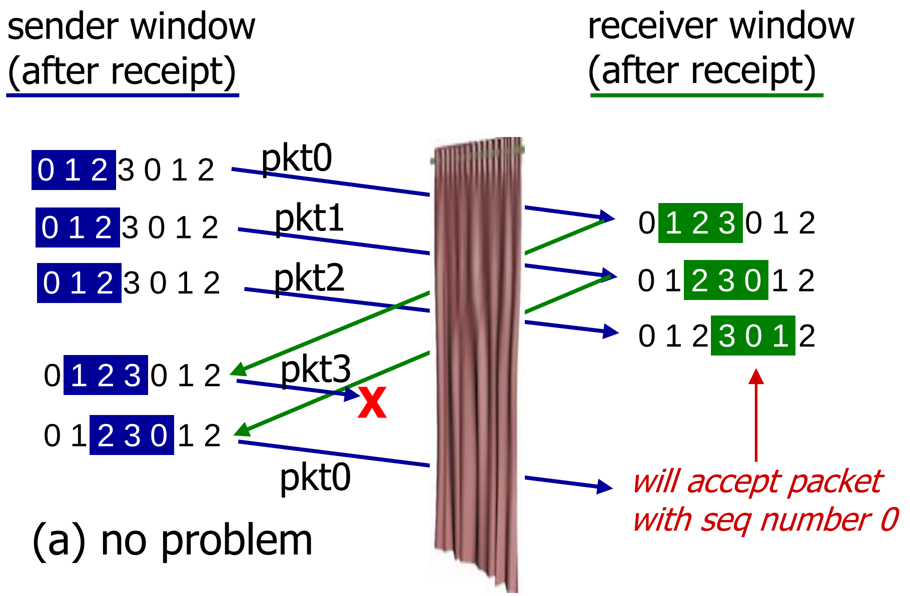
- out-of-order pkt:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

Selective repeat: dilemma

example:

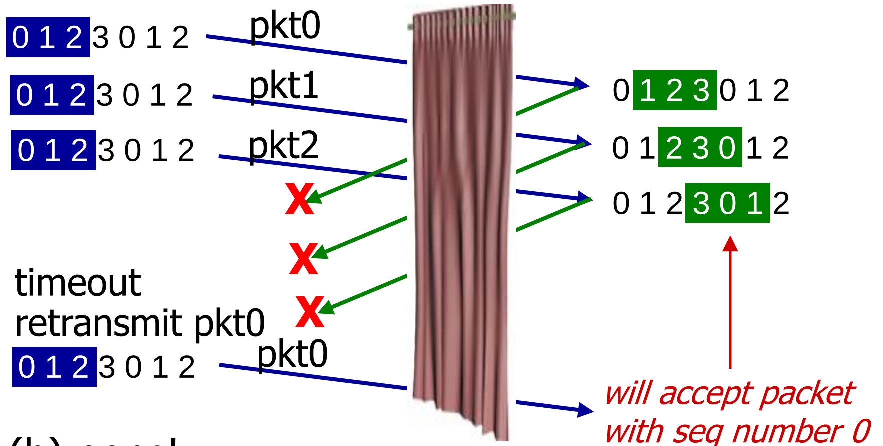
- seq #'s: 0, 1, 2, 3
- window size=3
  - receiver sees no difference in two scenarios!
  - duplicate data accepted as new in (b)

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



(a) no problem

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



(b) oops!

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

Namah Shivaya