

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

Performance of rdt3.0

- rdt3.0 is correct, but performance stinks
- e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

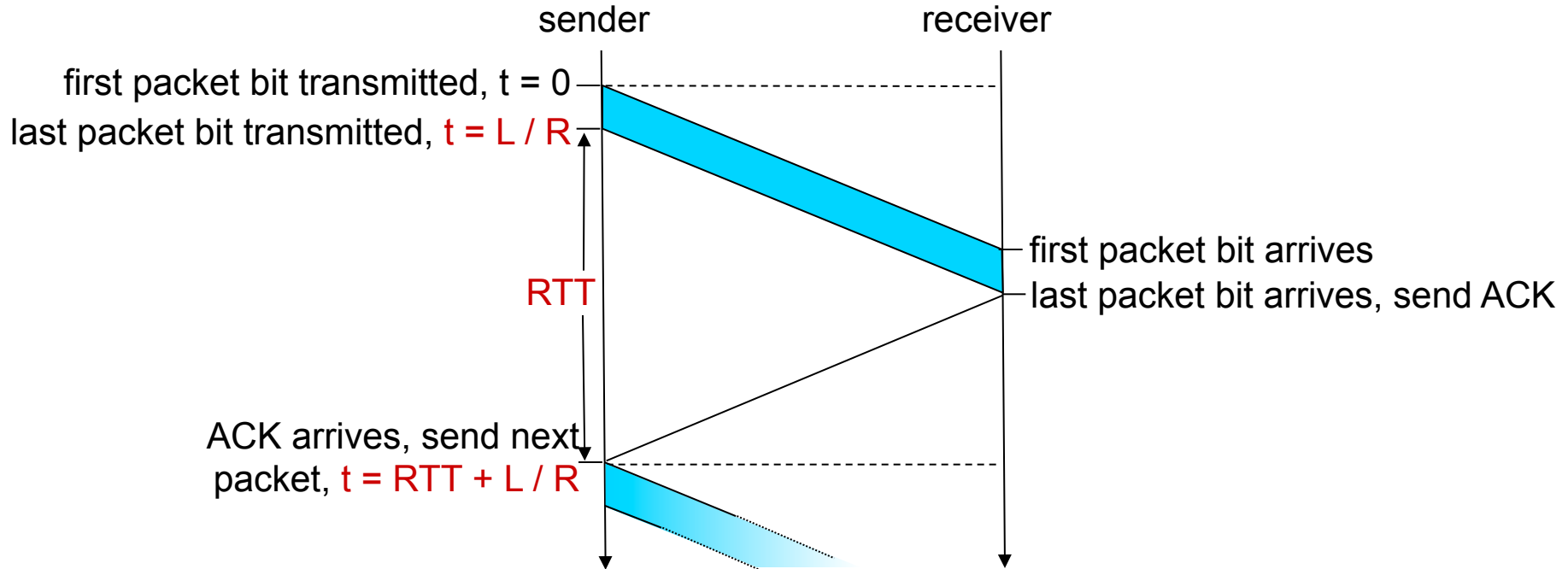
$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$

- U_{sender} : **utilization** – fraction of time sender busy sending

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

- if RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!

rdt3.0: stop-and-wait operation

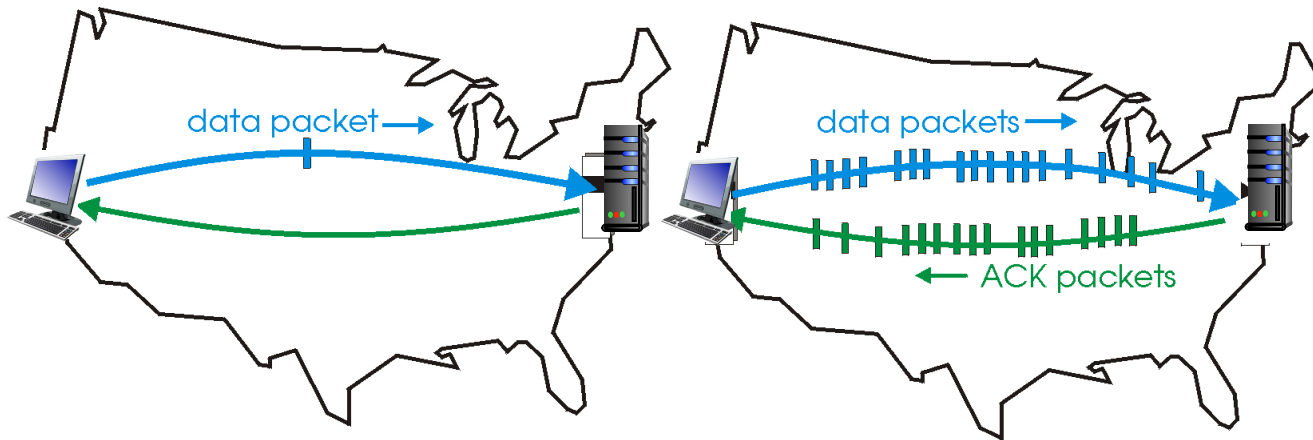


$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

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

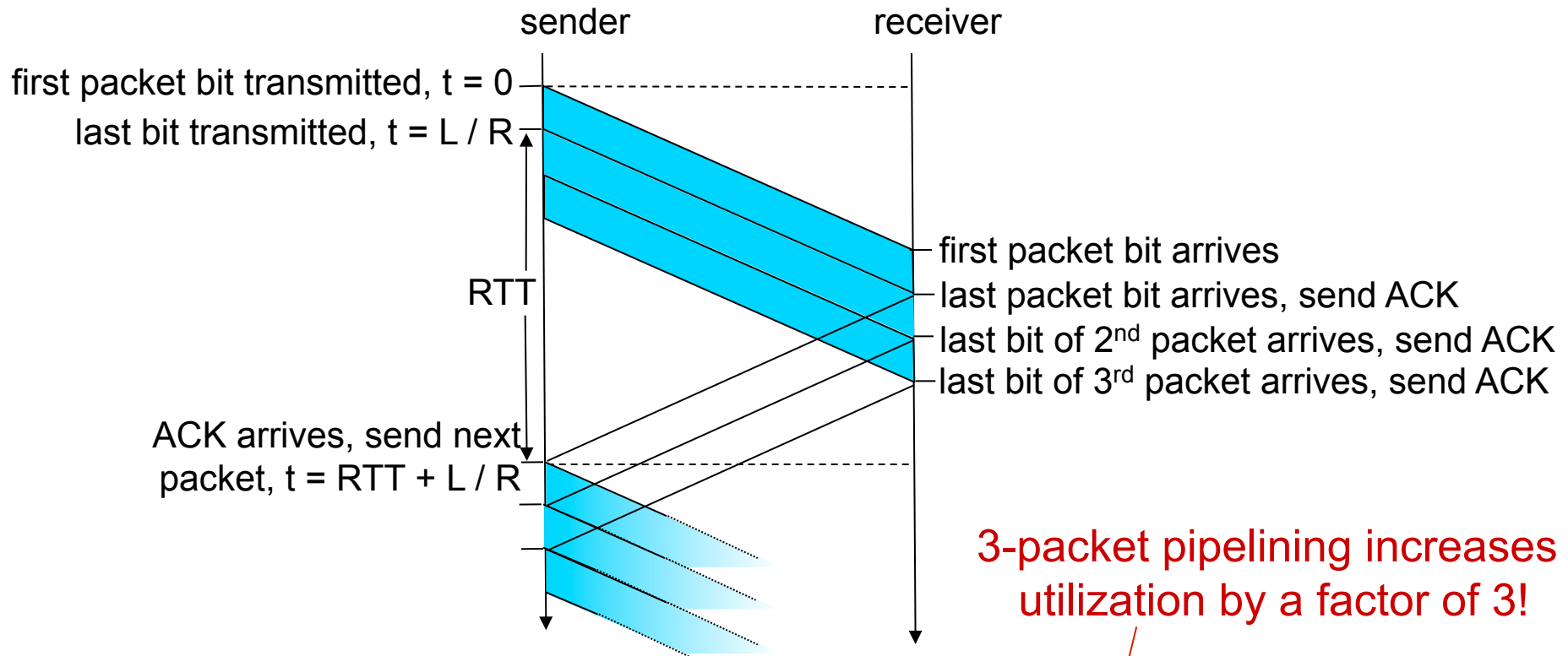


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

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

Pipelining: increased utilization



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: overview

Go-back-N:

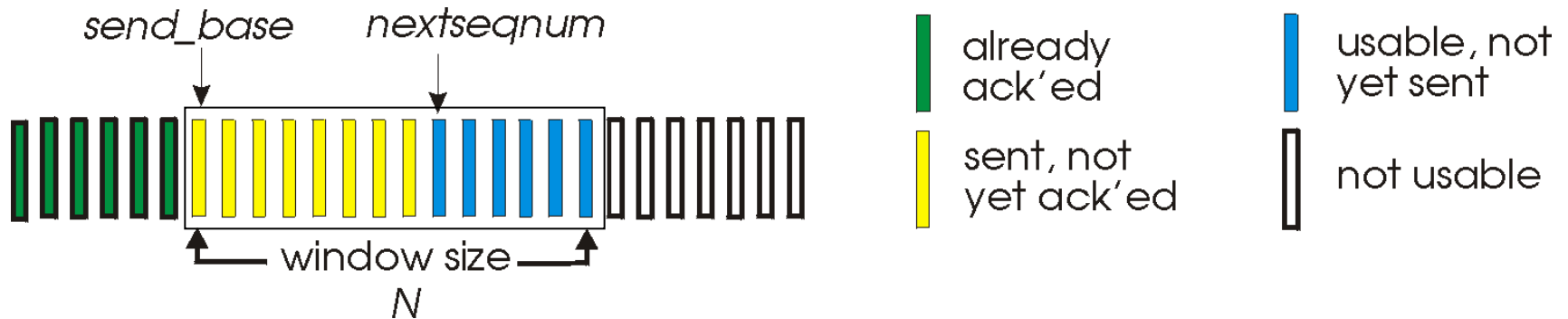
- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
 - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
 - when timer expires, retransmit *all* unacked packets

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

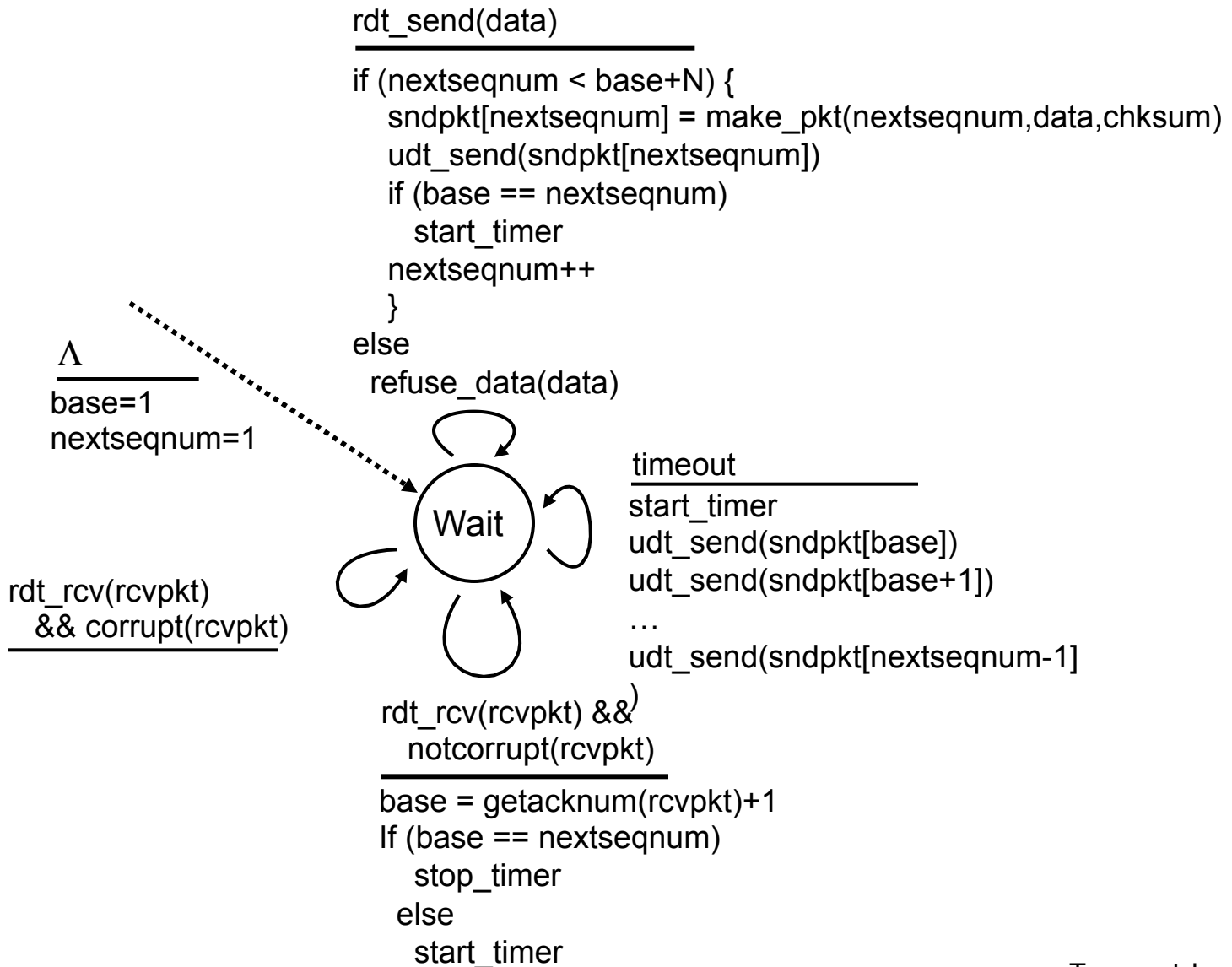
Go-Back-N: sender

- k-bit seq # in pkt header
- “window” of up to N, consecutive unack’ed pkts allowed

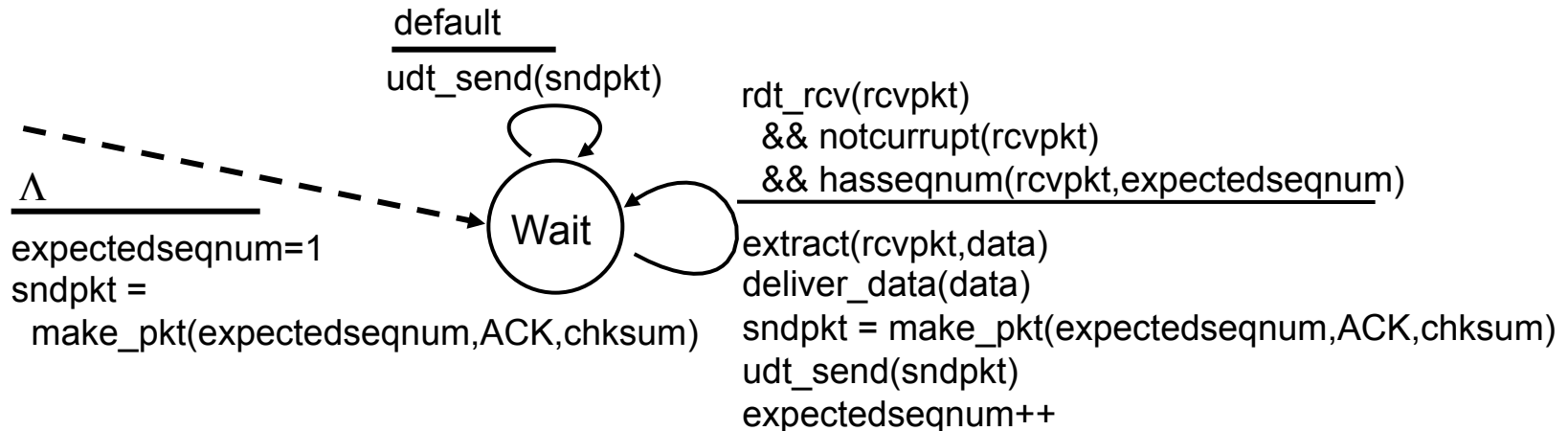


- ACK(n): ACKs all pkts up to, including seq # n - “cumulative ACK”
 - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- $timeout(n)$: retransmit packet n and all higher seq # pkts in window

GBN: sender extended FSM



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 #

GBN 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

ignore duplicate ACK



pkt 2 timeout

send pkt2
 send pkt3
 send pkt4
 send pkt5

receiver

receive pkt0, send ack0
 receive pkt1, send ack1

receive pkt3, discard,
 (re)send ack1

receive pkt4, discard,
 (re)send ack1

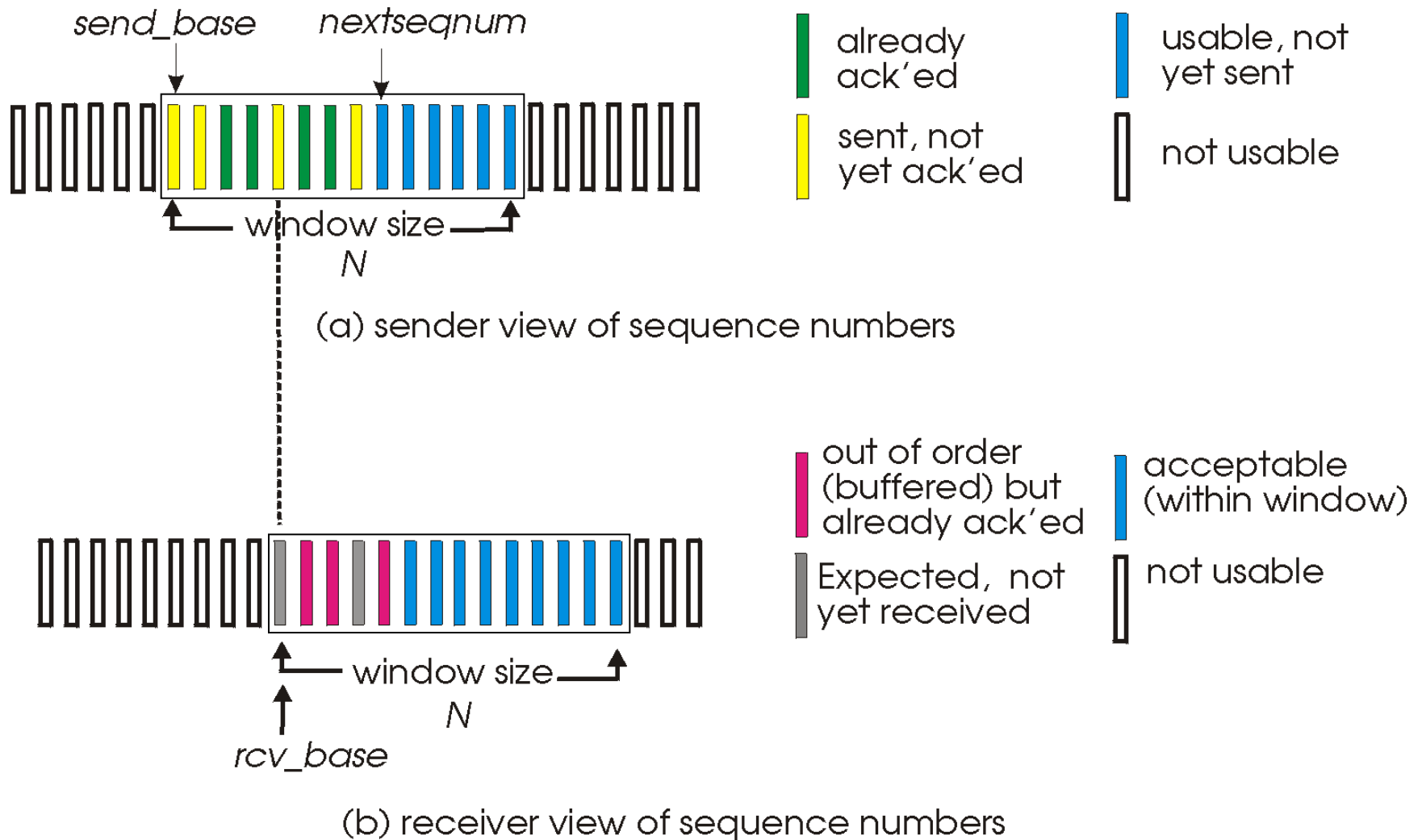
receive pkt5, discard,
 (re)send ack1

rcv pkt2, deliver, send ack2
 rcv pkt3, deliver, send ack3
 rcv pkt4, deliver, send ack4
 rcv pkt5, deliver, send ack5

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



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]

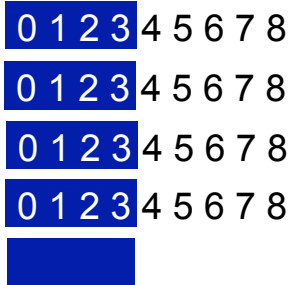
- ACK(n)

otherwise:

- ignore

Selective repeat in action

sender window (N=4)



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

Xloss

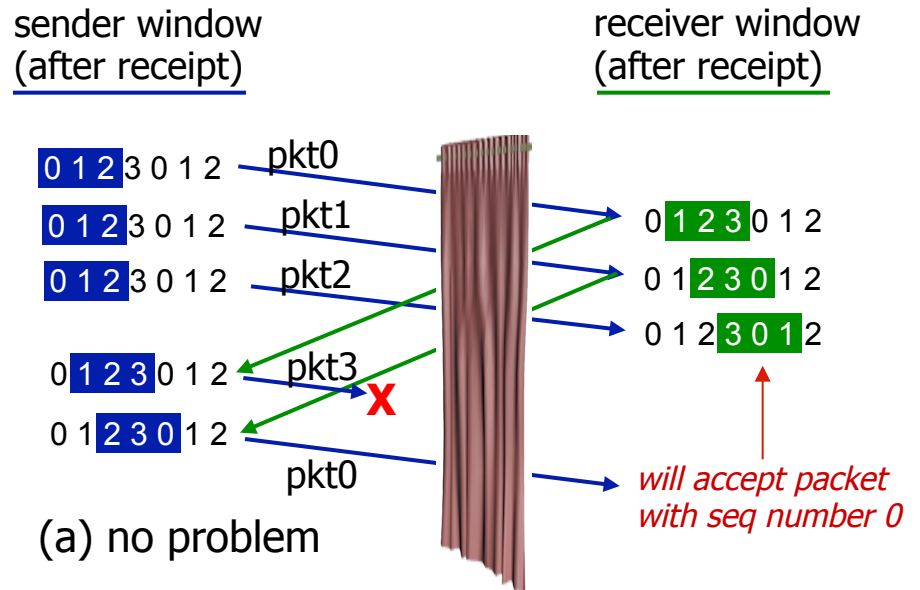
Q: what happens when ack2 arrives?

Selective repeat: dilemma

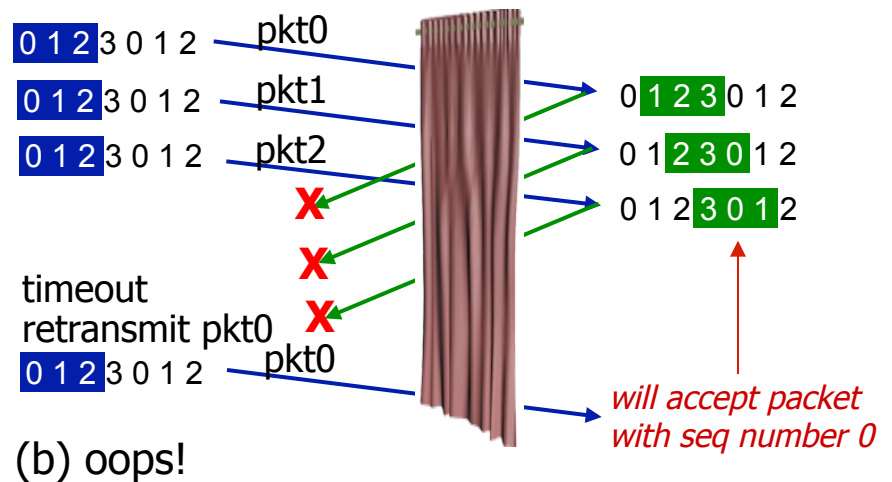
example:

- 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)?



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



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