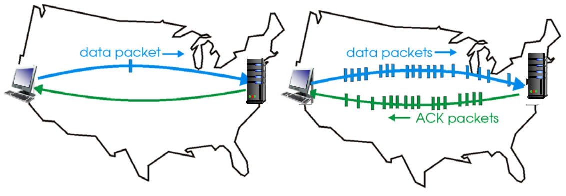
Pipelined rdt

- pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets
  - range of sequence numbers must be increased
  - buffering at sender and/or receiver



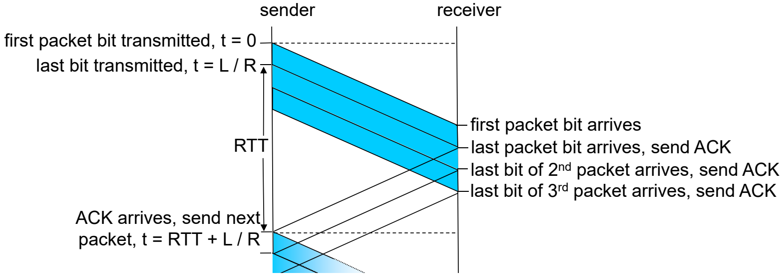
(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Pipelining can improve utilization

- example: 1Gbps link, 15ms propagation delay, 8000-bit packet

$$U_{\text{sender}} = \frac{3 \cdot \frac{L}{R}}{\frac{L}{R} + RTT} = \frac{3 \cdot 0.008}{30.008} = 0.00081$$

  - 3-packet pipelining improves utilization by a factor of 3.



sender

receiver

first packet bit transmitted, t = 0

last bit transmitted, t = L / R

RTT

first packet bit arrives

last packet bit arrives, send ACK

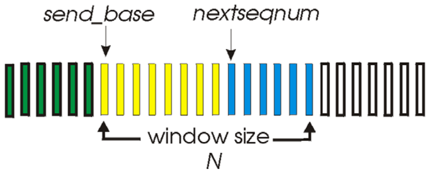
last bit of 2nd packet arrives, send ACK

last bit of 3rd packet arrives, send ACK

ACK arrives, send next packet, t = RTT + L / R

Go-Back-N: sender

- sender: "window" of up to *N* (consecutive sent but unACKed) pkts
  - k-bit seq # in pkt header



🟩 already ack'ed

🟨 sent, not yet ack'ed

🟦 usable, not yet sent

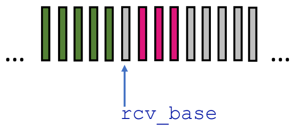
□ not usable

- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet *n* and all (available) packets in window

Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of in-order packet:
  - update `rcv_base`
- on receipt of out-of-order packet:
  - discard (don't buffer) or buffer the packet: an implementation decision
  - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:

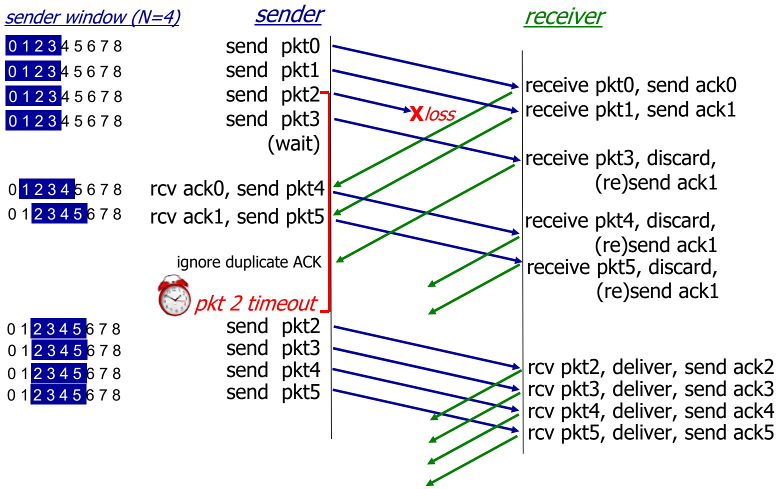


🟩 received and ACKed

🟥 Out-of-order: received but not ACKed

▯ Not received

Go-Back-N 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

X loss

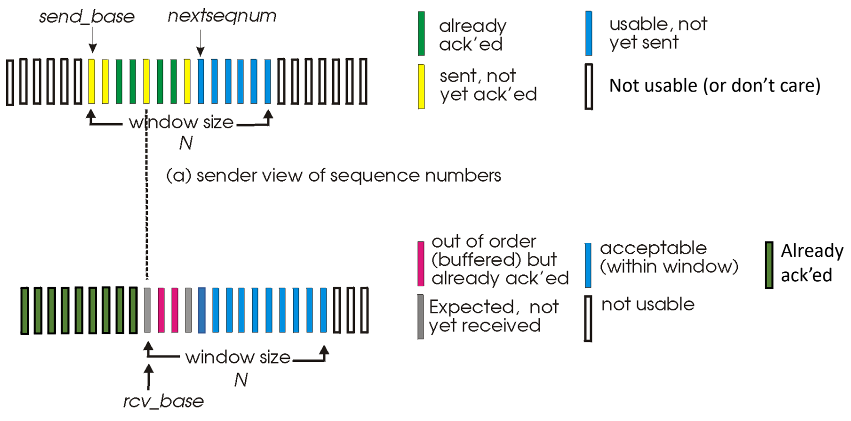
receiver

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 packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
  - sender maintains timer for each unACKed pkt
- sender window
  - *N* consecutive seq #
  - limits # of sent and still unACKed packets

Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

(b) receiver view of sequence numbers

Selective repeat: sender and receiver

sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend packet *n*, restart timer

ACK(*n*) in [sendbase, sendbase+N-1]:

- mark packet *n* as received
- if *n* is smallest unACKed packet, advance window base to next unACKed seq #

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

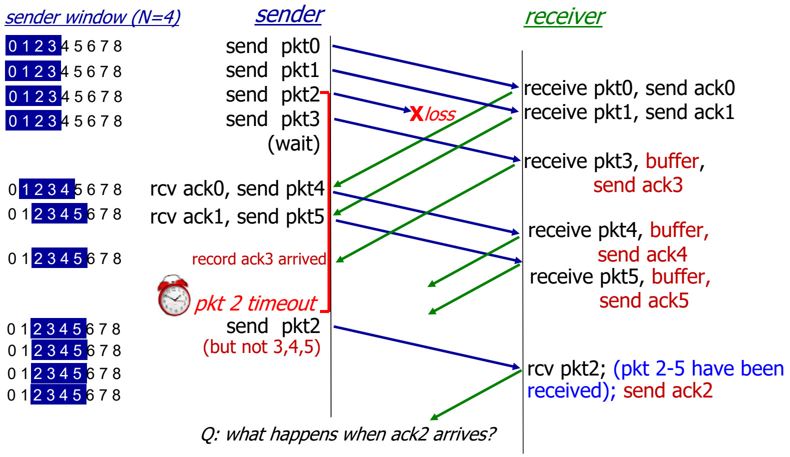
packet *n* in [rcvbase-N, rcvbase-1]

- send ACK(*n*) back to the sender

otherwise:

- ignore

Selective Repeat in action



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)

loss

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, buffer, send ack3

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, buffer, send ack4

receive pkt5, buffer, send ack5

0 1 2 3 4 5 6 7 8 record ack3 arrived

pkt 2 timeout

0 1 2 3 4 5 6 7 8 send pkt2 (but not 3,4,5)

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

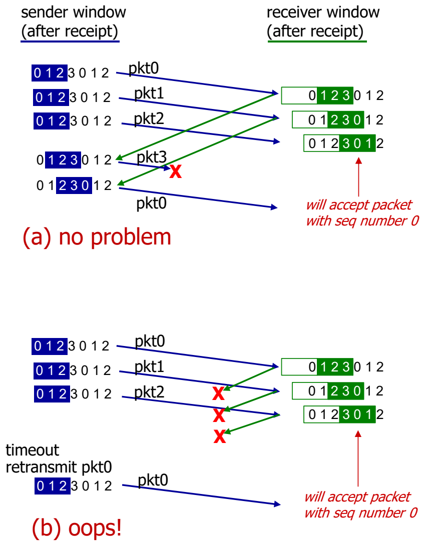
rcv pkt2; (pkt 2-5 have been received); send ack2

Q: what happens when ack2 arrives?

Selective repeat: a dilemma!

example:

- use 2-bit seq # : {0, 1, 2, 3}
- window size = 3



sender window (after receipt)

receiver window (after receipt)

0 1 2 3 0 1 2 pkt0

0 1 2 3 0 1 2 pkt1

0 1 2 3 0 1 2 pkt2

0 1 2 3 0 1 2

0 1 2 3 0 1 2

0 1 2 3 0 1 2 pkt3

0 1 2 3 0 1 2

pkt0

will accept packet with seq number 0

(a) no problem

0 1 2 3 0 1 2 pkt0

0 1 2 3 0 1 2 pkt1

0 1 2 3 0 1 2 pkt2

0 1 2 3 0 1 2

0 1 2 3 0 1 2

0 1 2 3 0 1 2

timeout retransmit pkt0

0 1 2 3 0 1 2 pkt0

will accept packet with seq number 0

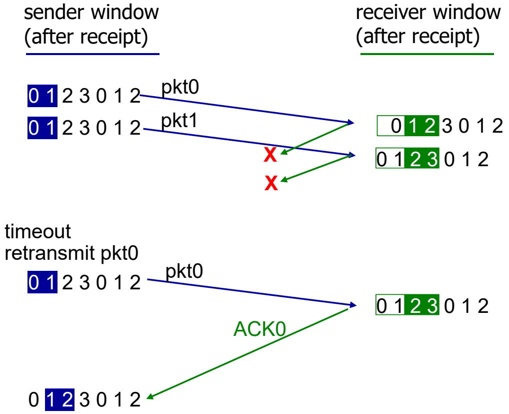
(b) oops!

Selective repeat: a dilemma!

example:

- use 2-bit seq # : {0, 1, 2, 3}
- window size = 2

Q: what relationship is needed between sequence # size and window size to avoid the problem in scenario (b)?



sender window (after receipt)

receiver window (after receipt)

0 1 2 3 0 1 2 pkt0

0 1 2 3 0 1 2 pkt1

0 1 2 3 0 1 2

0 1 2 3 0 1 2

timeout retransmit pkt0

0 1 2 3 0 1 2 pkt0

0 1 2 3 0 1 2

ACK0

0 1 2 3 0 1 2