The Transport Layer
Reliability

CS 352, Lecture 7

http://www.cs.rutgers.edu/~sn624/352-S19

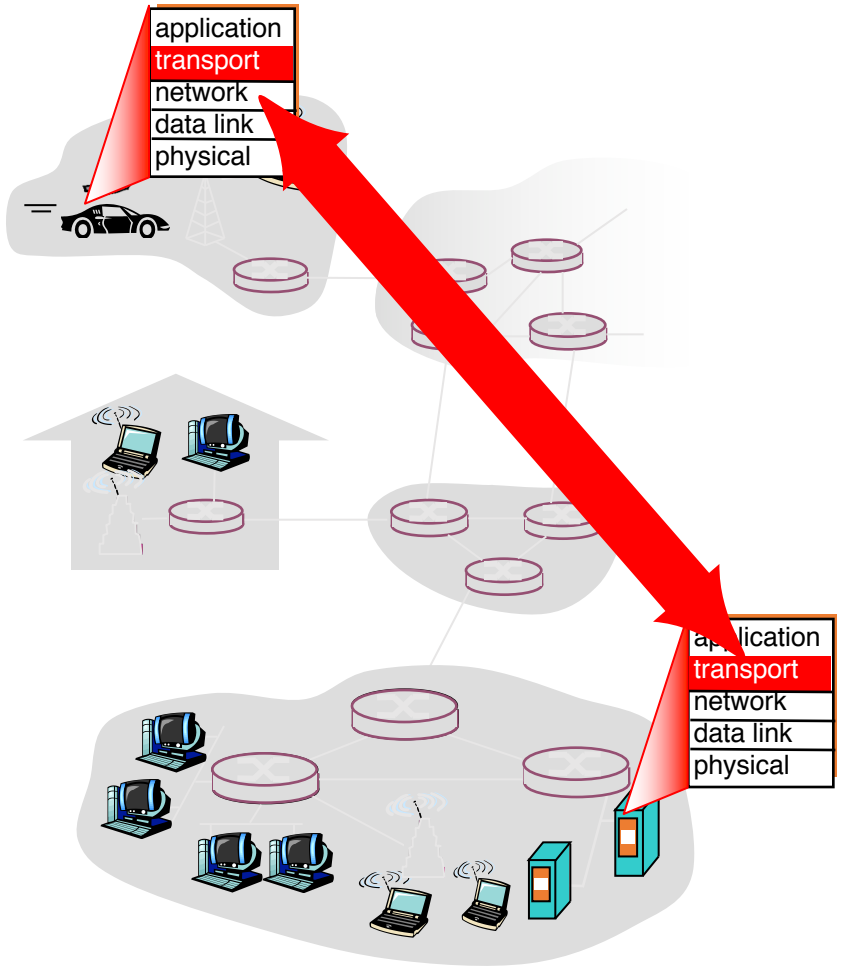
Srinivas Narayana

(slides heavily adapted from text authors' material)

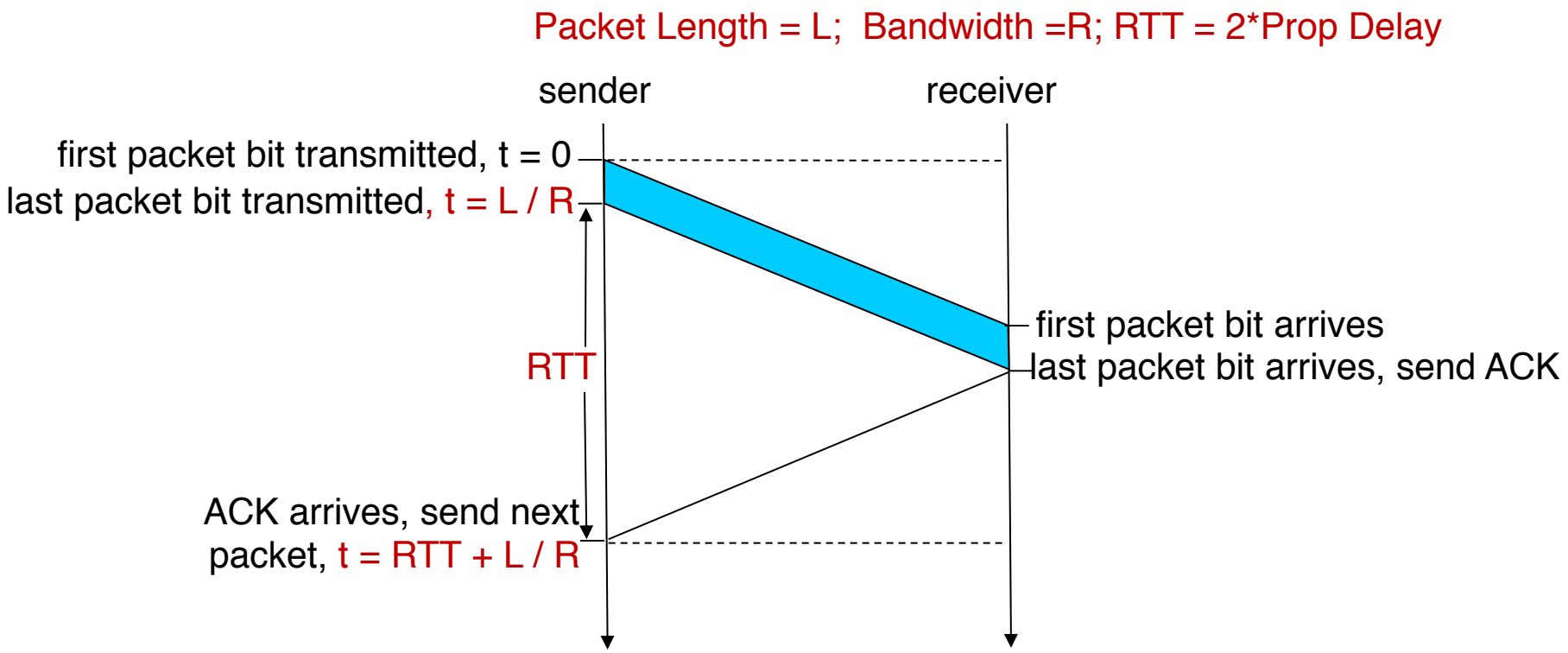
RUTGERS
UNIVERSITY | NEW BRUNSWICK

Quick recap: Transport

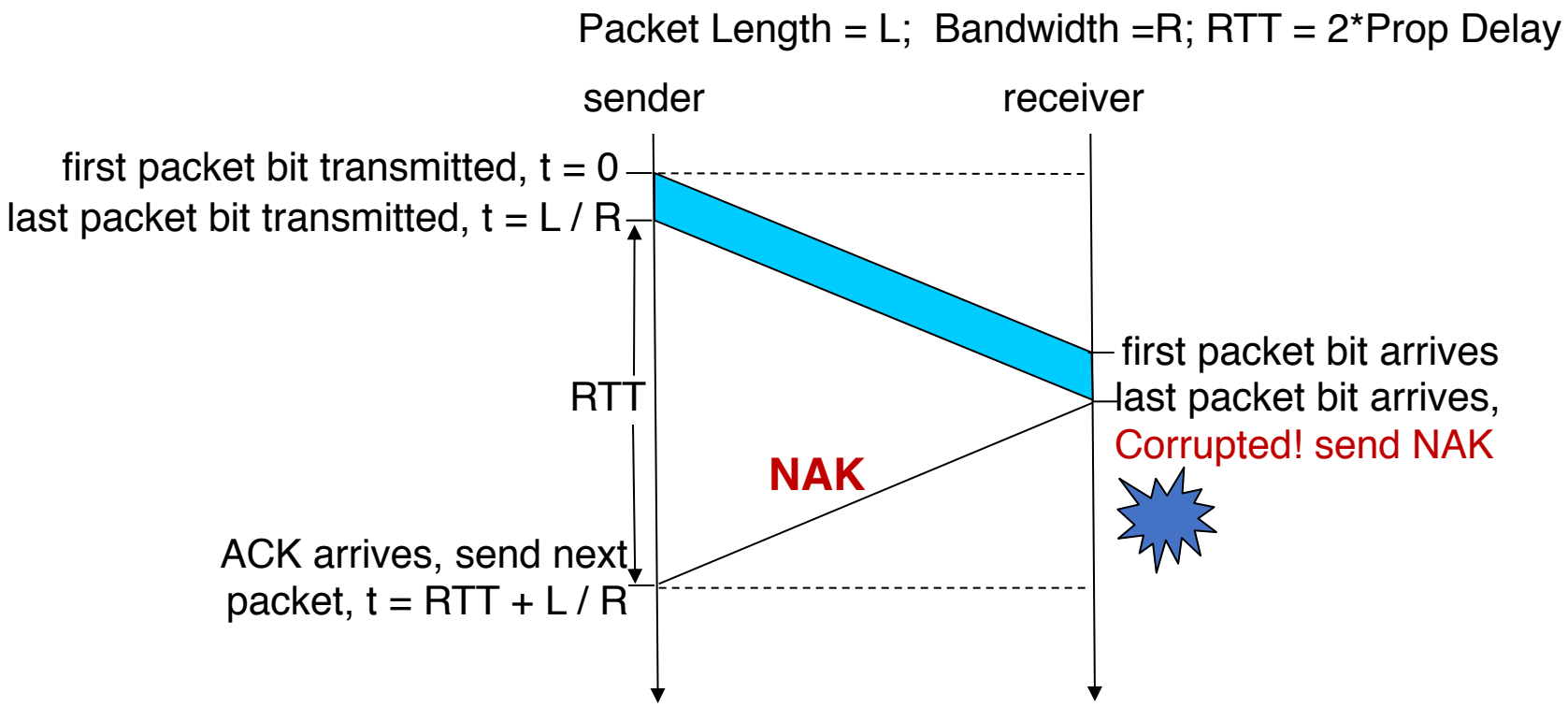
- Provide logical communication between app processes running on different hosts
- Providing reliable communication:
  - “Stop and wait” protocols
- Concepts:
  - Acknowledgements (ACKs)
  - Timeouts
  - Retransmission
  - Sequence numbers



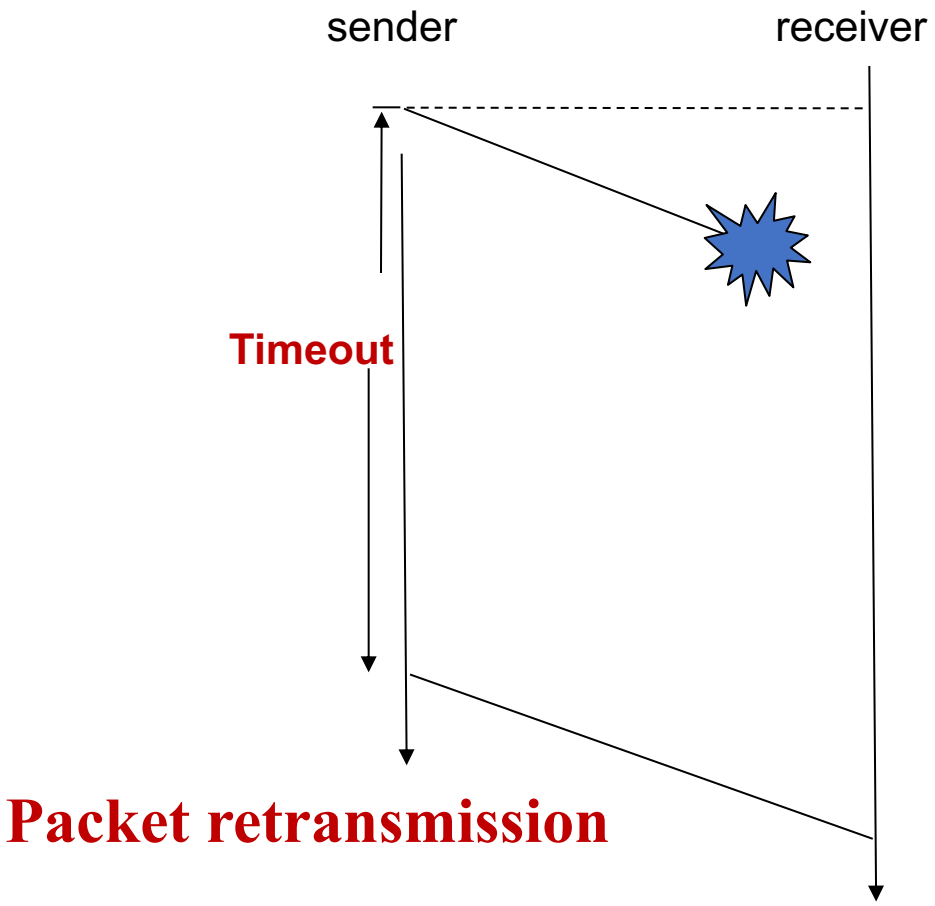
Stop-and-wait: normal operation



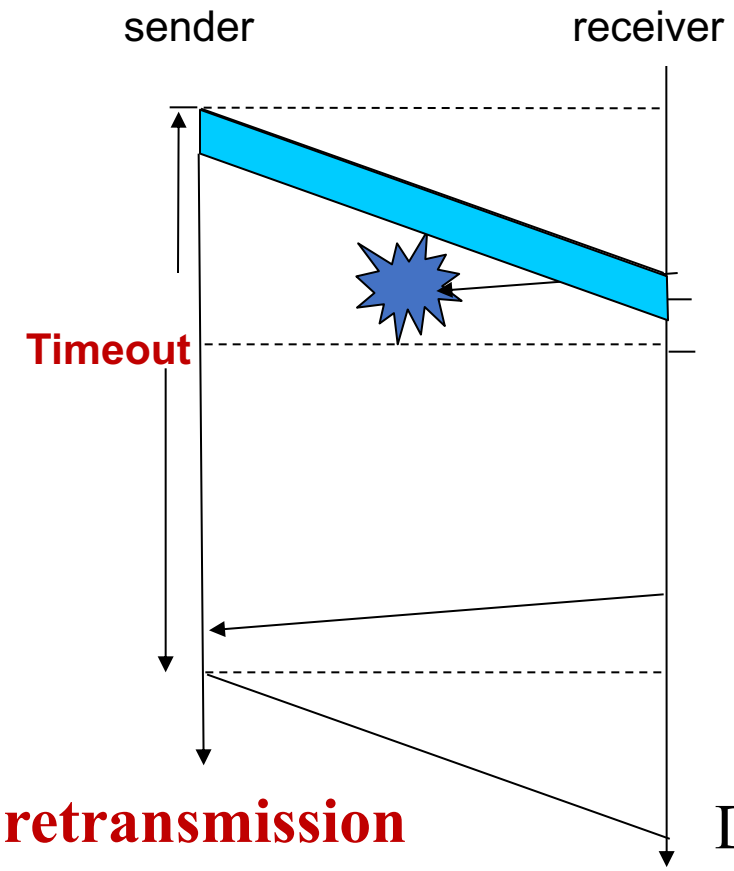
Stop-and-wait: packet corrupted



Stop-and-wait: packet lost



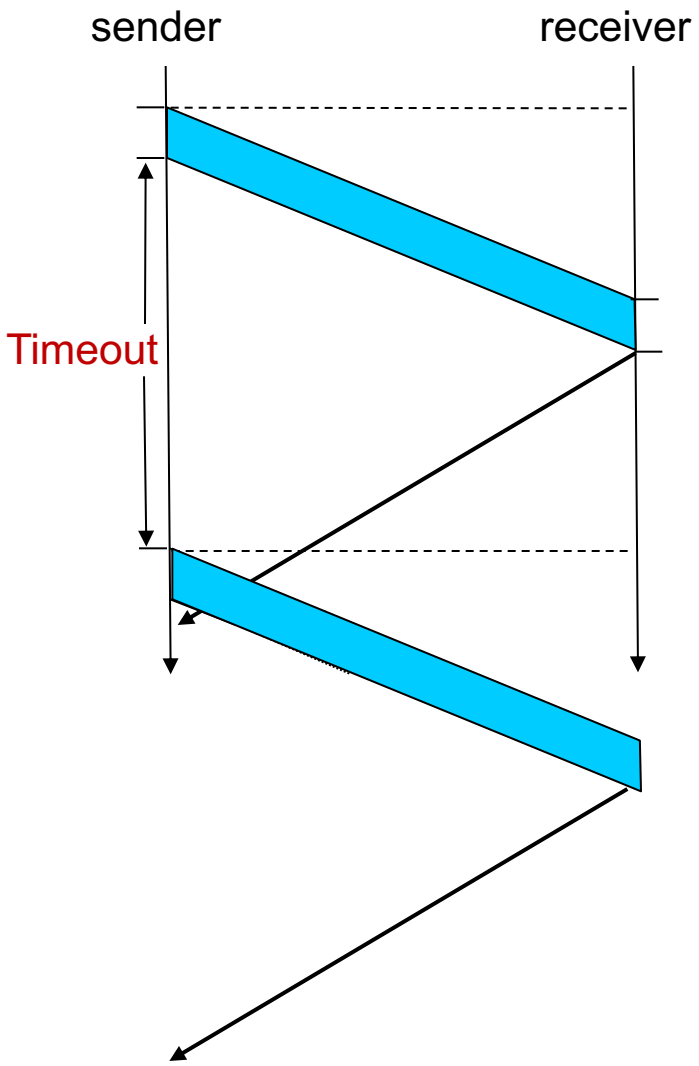
Stop-and-wait: ACK lost!



Packet retransmission

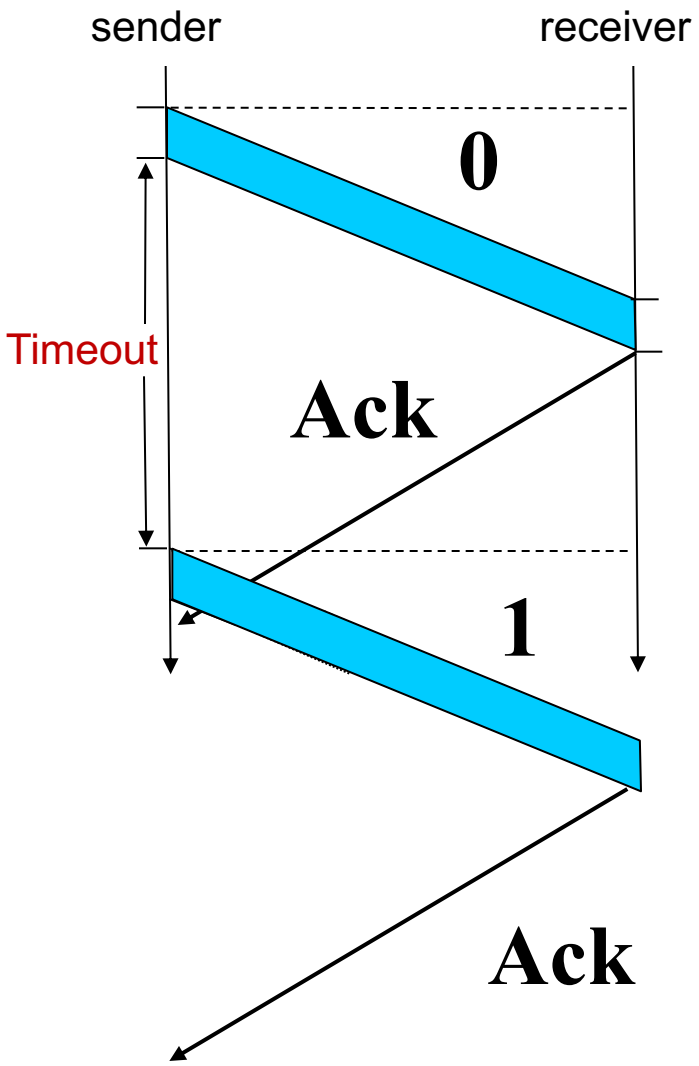
Duplicate transmission received

Stop-and-wait: ACKs may be delayed!



Timeout too short
Duplicate Transmission

Stop-and-wait: Sequence numbers

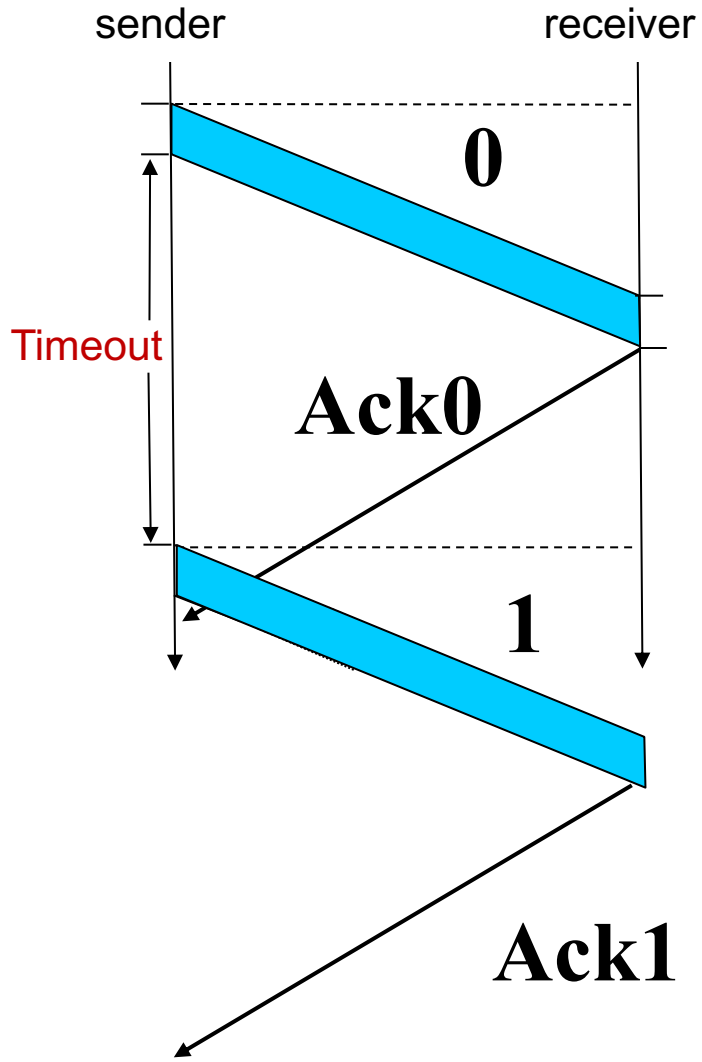


Sequence numbers help detect duplicate transmissions at the receiver.

Often, protocols just use ACKs (no NAKs)

- Instead of NAK, receiver sends ACK for last packet received OK
  - receiver must *explicitly* include seq # of pkt being ACKed

- Duplicate ACK at sender results in same action as NAK: *retransmit current packet*



Discussion of stop and wait protocols

sender:

- seq # added to pkt
- two seq. #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted

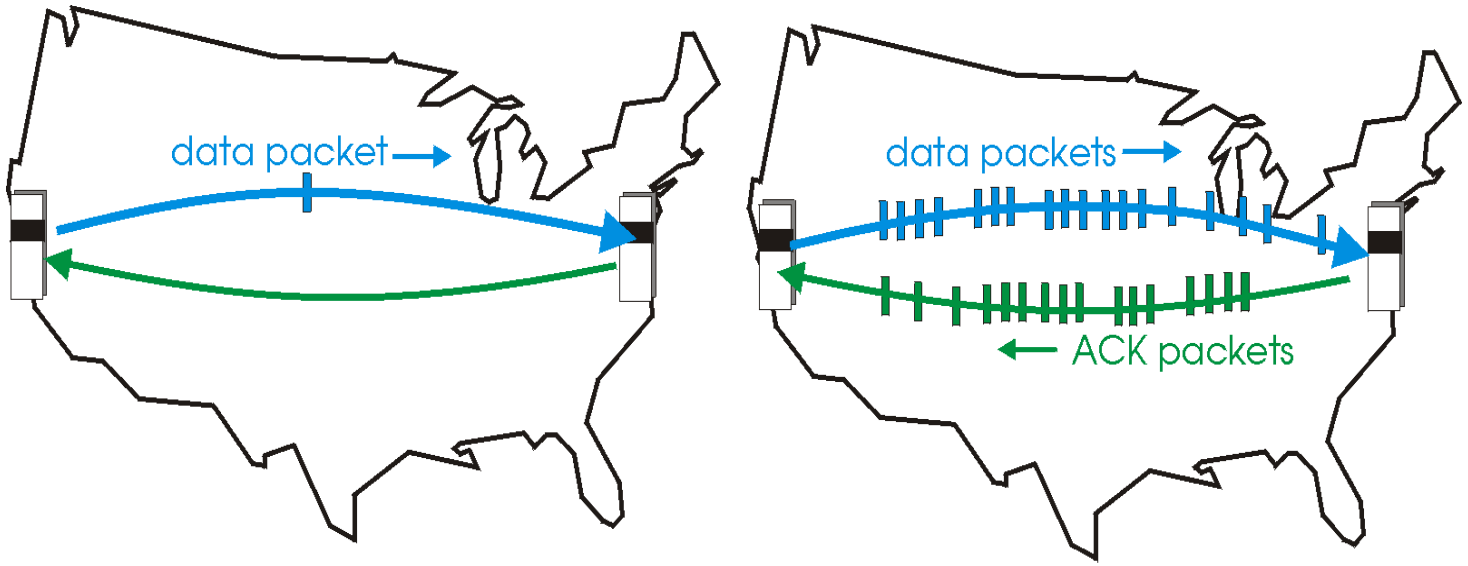
receiver:

- must check if received packet is duplicate
  - ACK indicates received packet, but also next packet expected
- note: receiver *cannot* know if its last ACK/NAK received OK at sender

Problem with stop and wait: low utilization of available capacity
Reason: Transmitting one packet per round-trip time

Pipelined protocols

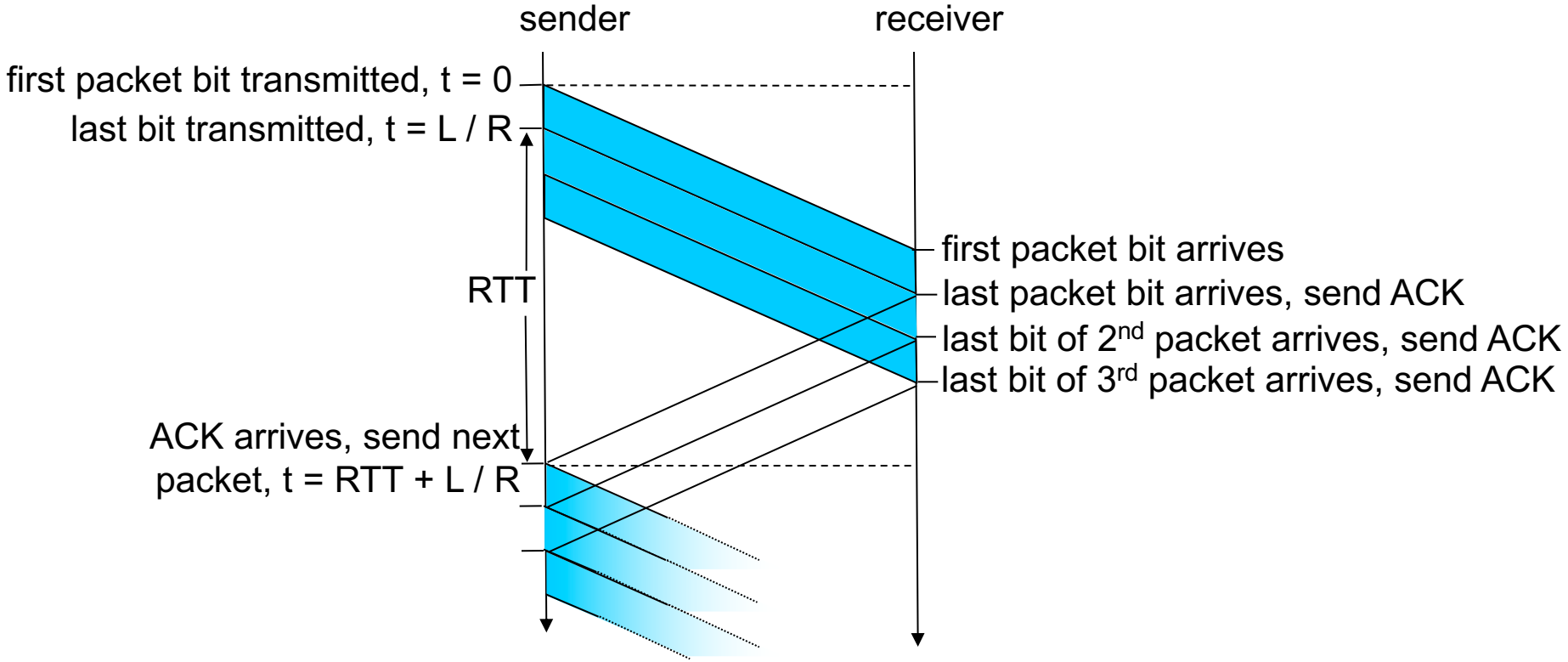
Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts



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

(b) a pipelined protocol in operation

Pipelining Example: increased utilization



Sliding Window Protocols: Definitions

Window: informally, a subset of packets the sender or receiver consider “in flight”

Sequence Number: Each frame is assigned a sequence number that is incremented as each frame is transmitted

Sender Window: Keeps track of sequence numbers for frames that have been sent but not yet acknowledged

Sender Window size: The maximum number of frames the sender may transmit without receiving any acknowledgements

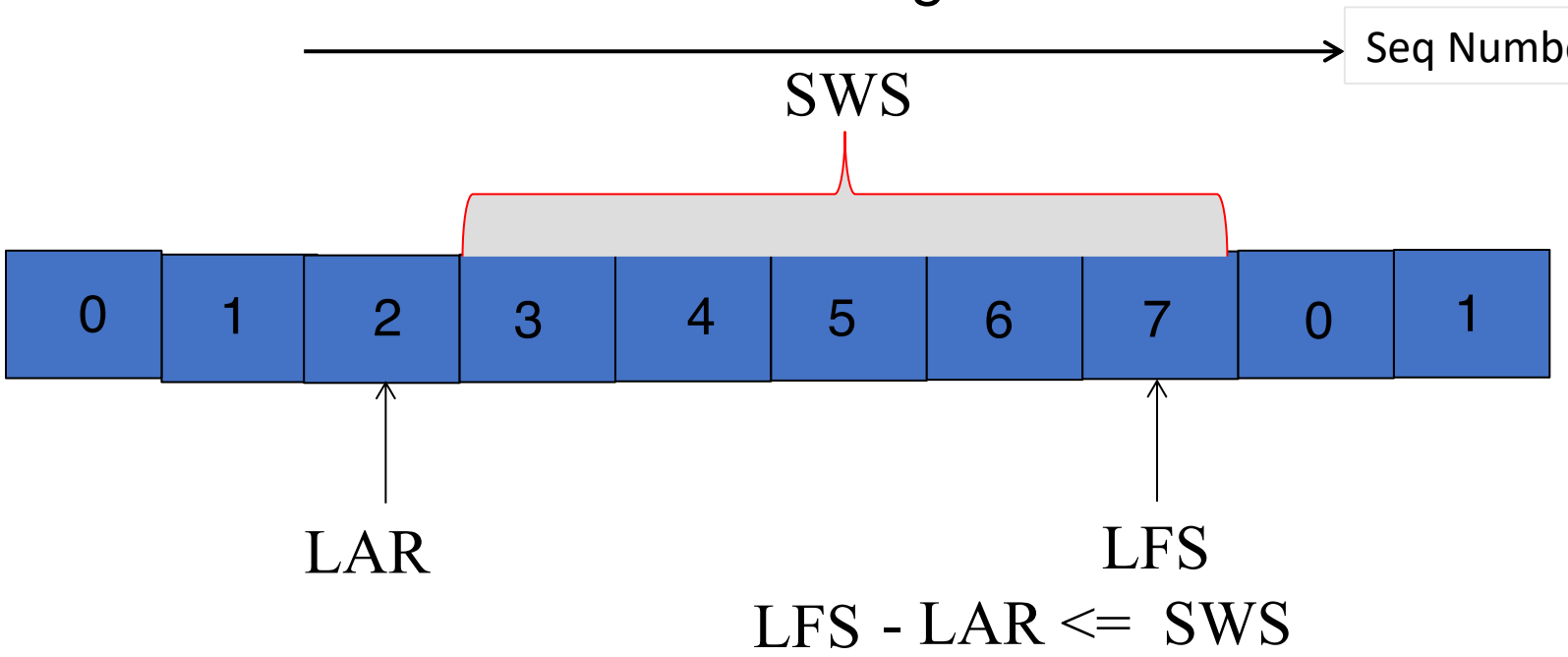
Last Frame sent: The highest sequence number frames sent

Last Ack received: sequence number of last ACK

Sliding Window with Maximum Sender Window Size *SWS*

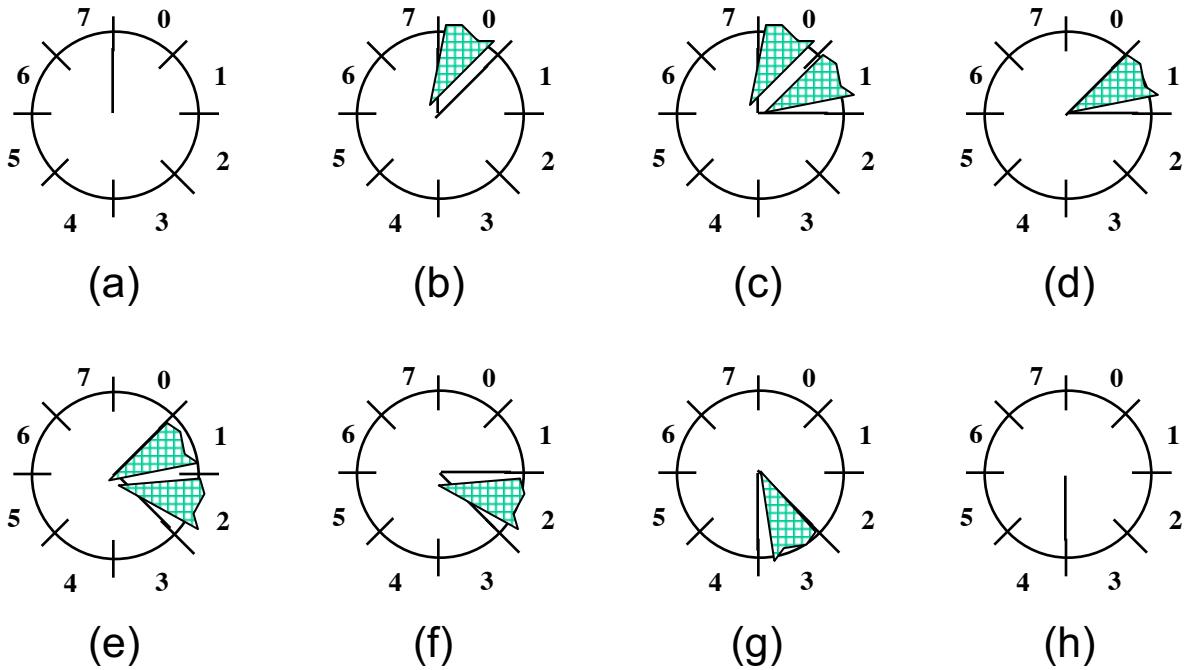
With a maximum window size of *SWS*, the sender can transmit up to *SWS* frames before stopping and waiting for ACKs

This allows the sender to transmit several frames before waiting for an acknowledgement



Sequence numbers wrap around! (limited number of bits in header)

Sender-Side Window with W_S=2



(a) Initial window state	(e) Send frame 2
(b) Send frame 0	(f) ACK for frame 1 arrives
(c) Send frame 1	(g) ACK for frame 2 arrives, send frame 3
(d) ACK for frame 0 arrives	(h) ACK for frame 3 arrives

Sliding Window Protocols
Receiver Definitions

Receiver Window: Keeps track of sequence numbers for frames the receiver is allowed to accept

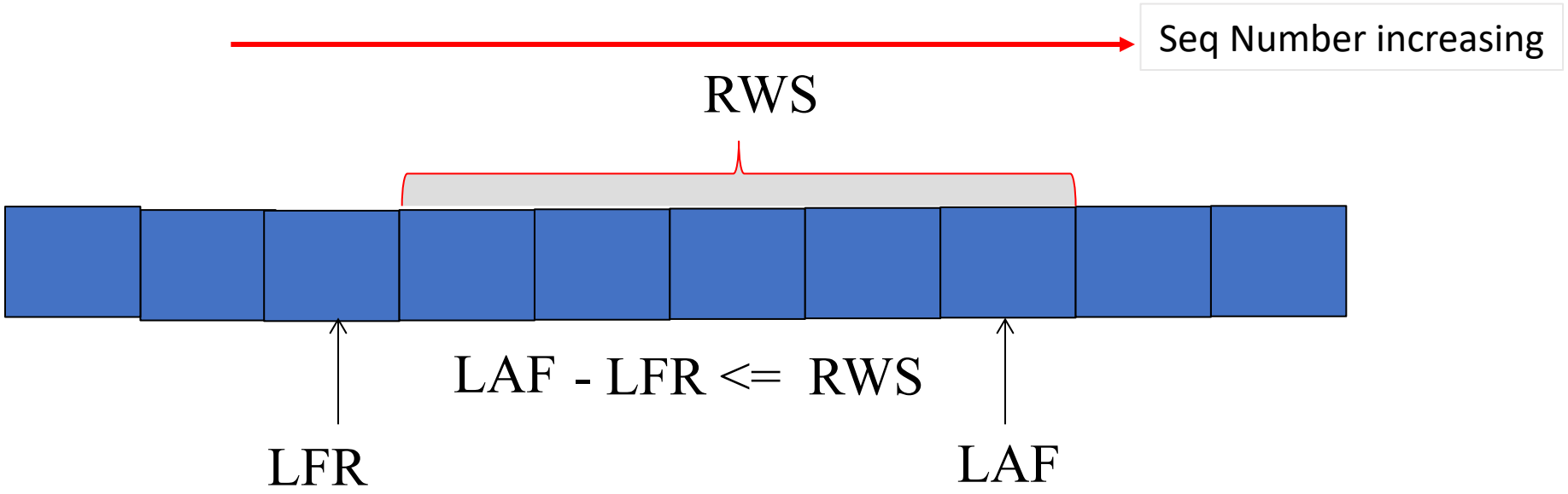
Receiver Window size: The maximum number of frames the receiver may receive before returning an acknowledgement to the sender

Largest Frame received: The highest in-sequence numbered frame received

Largest Acceptable Frame: Highest sequence number acceptable

Sliding Window with Maximum Receiver Window Size *RWS*

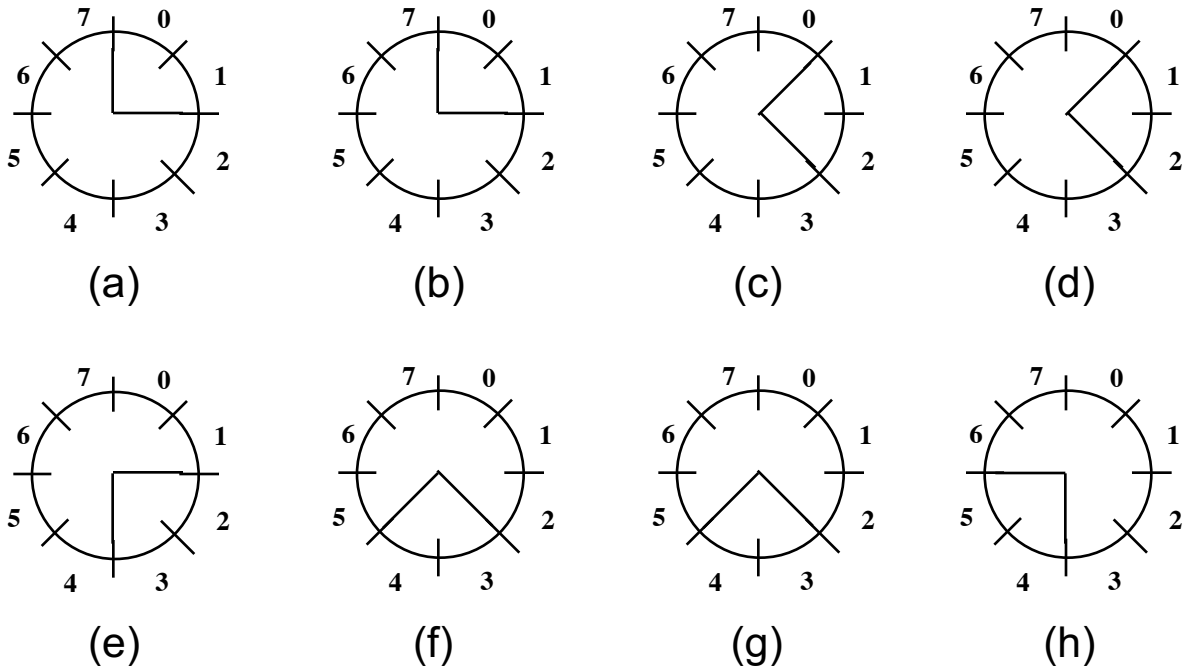
With window *RWS*, receiver rejects pkts if SeqNum <= LFR or SeqNum > LAF



Receiver acknowledges seqnumber s.t all earlier frames have been received

Cumulative ACK

Receiver-Side Window with W_R=2



(a) Initial window state	(e) Frame 1 arrives, ACK frame 1
(b) Nothing happens	(f) Frame 2 arrives, ACK frame 2
(c) Frame 0 arrives, ACK frame 0	(g) Nothing happens
(d) Nothing happens	(h) Frame 3 arrives, ACK frame 3

What about Errors?

What if a data or acknowledgement frame is lost when using a sliding window protocol?

Two Solutions:

Go Back N

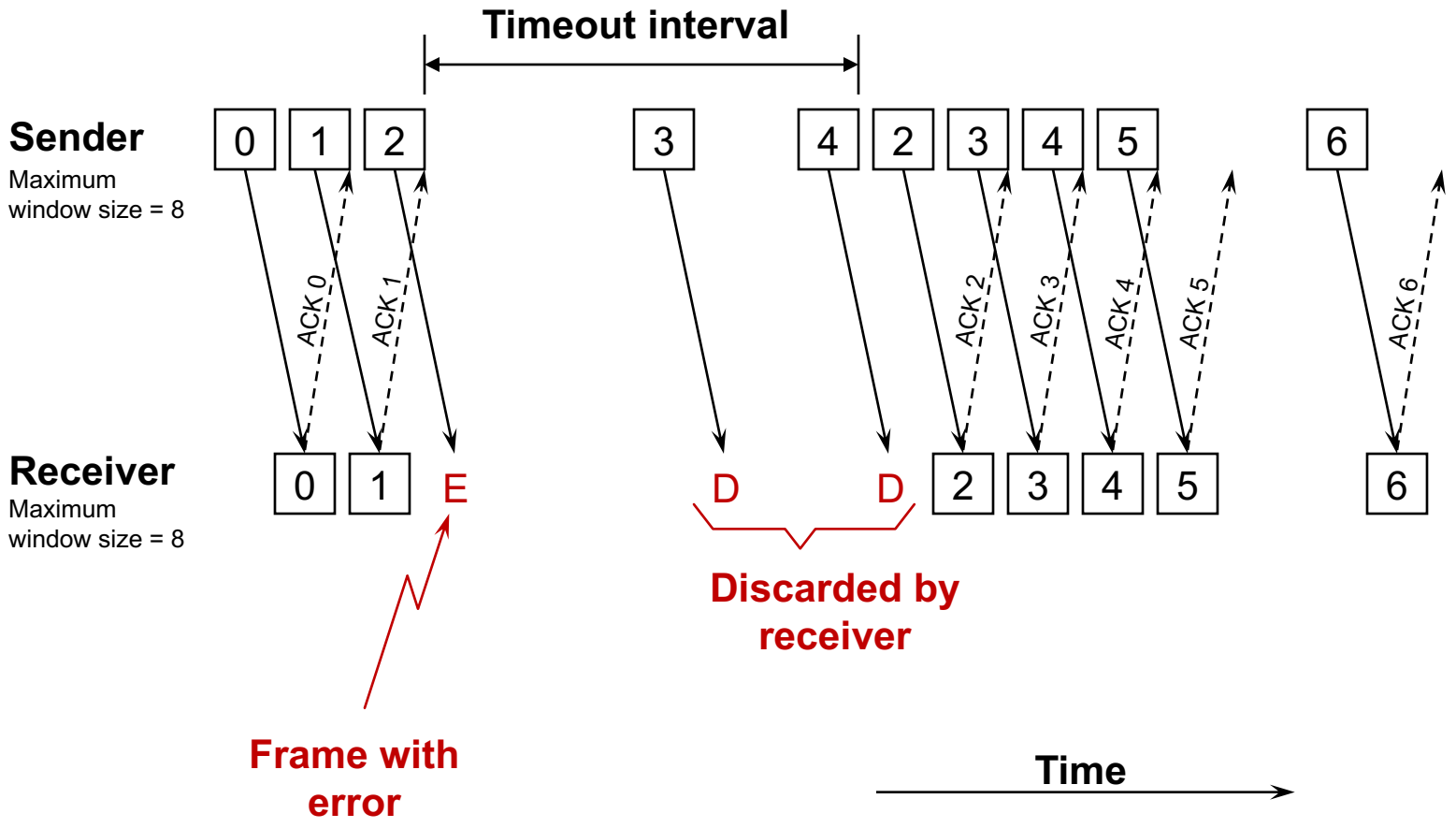
Selective Repeat

Sliding Window with Go Back N

- When the receiver notices a missing or erroneous frame, it simply discards all frames with greater sequence numbers and sends no ACK

- The sender will eventually time out and retransmit all the frames in its sending window

Go Back N


Timeout interval
Sender
Maximum
window size = 8
0
1
2
3
4
2
3
4
5
6
ACK 0
ACK 1
ACK 2
ACK 3
ACK 4
ACK 5
ACK 6
Receiver
Maximum
window size = 8
0
1
E
D
D
2
3
4
5
6
Discarded by
receiver
Frame with
error
Time

Go Back N *(cont'd)*

Go Back N can recover from erroneous or missing frames

But…

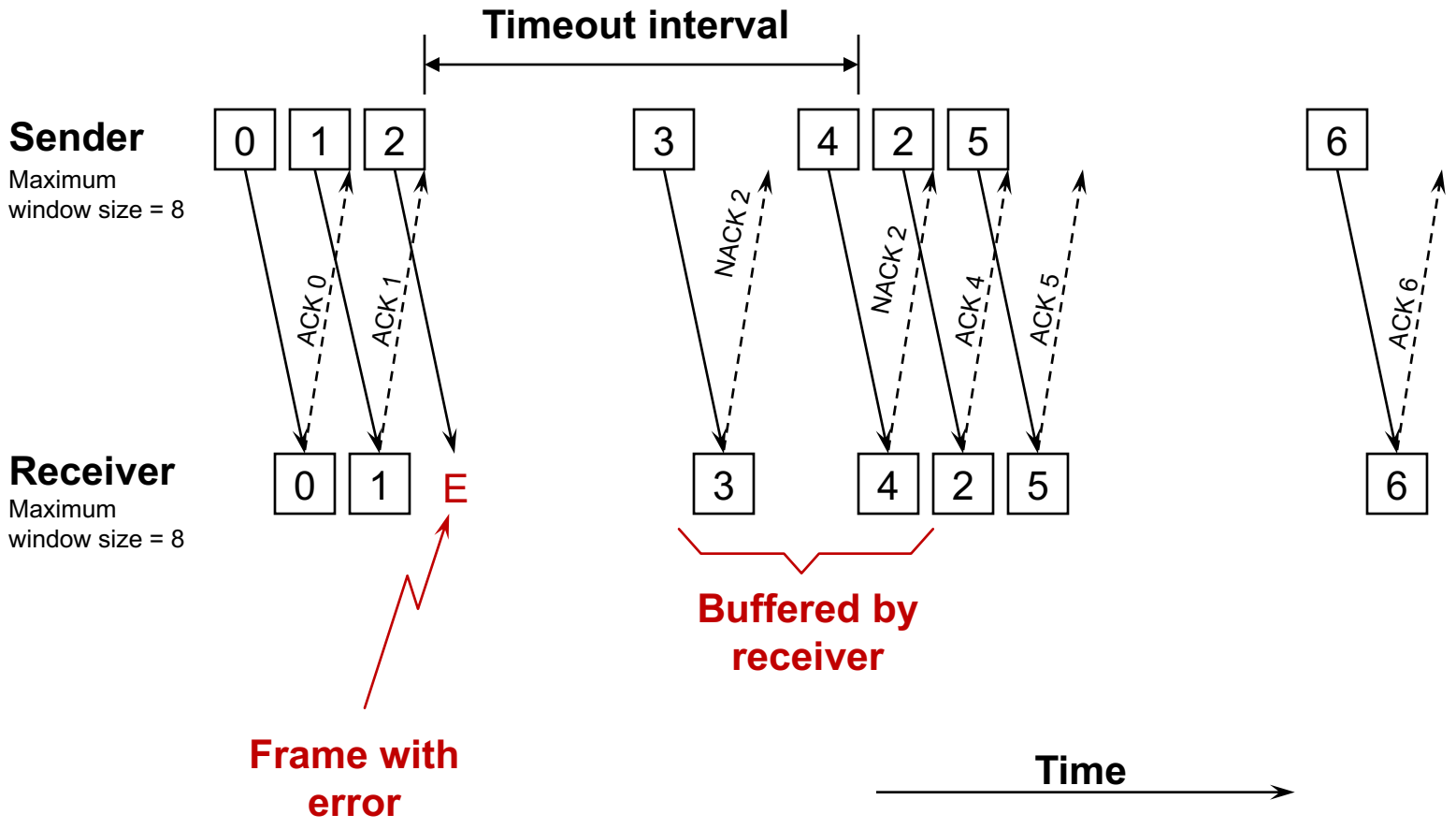
It is wasteful. If there are errors, the sender will spend time retransmitting frames the receiver has already seen

Sliding Window with Selective Repeat

Idea: sender should only retransmit dropped/corrupted segments.

- The receiver stores all the correct frames that arrive following the bad one. (Note that the receiver requires a memory buffer for each sequence number in its receiver window.)
- When the receiver notices a skipped sequence number, it keeps acknowledging the last good sequence number (cumulative ACK)
- When the sender times out waiting for an acknowledgement, it just retransmits the one unacknowledged frame, not all its successors.

Selective Repeat



Pipelined protocols: overview

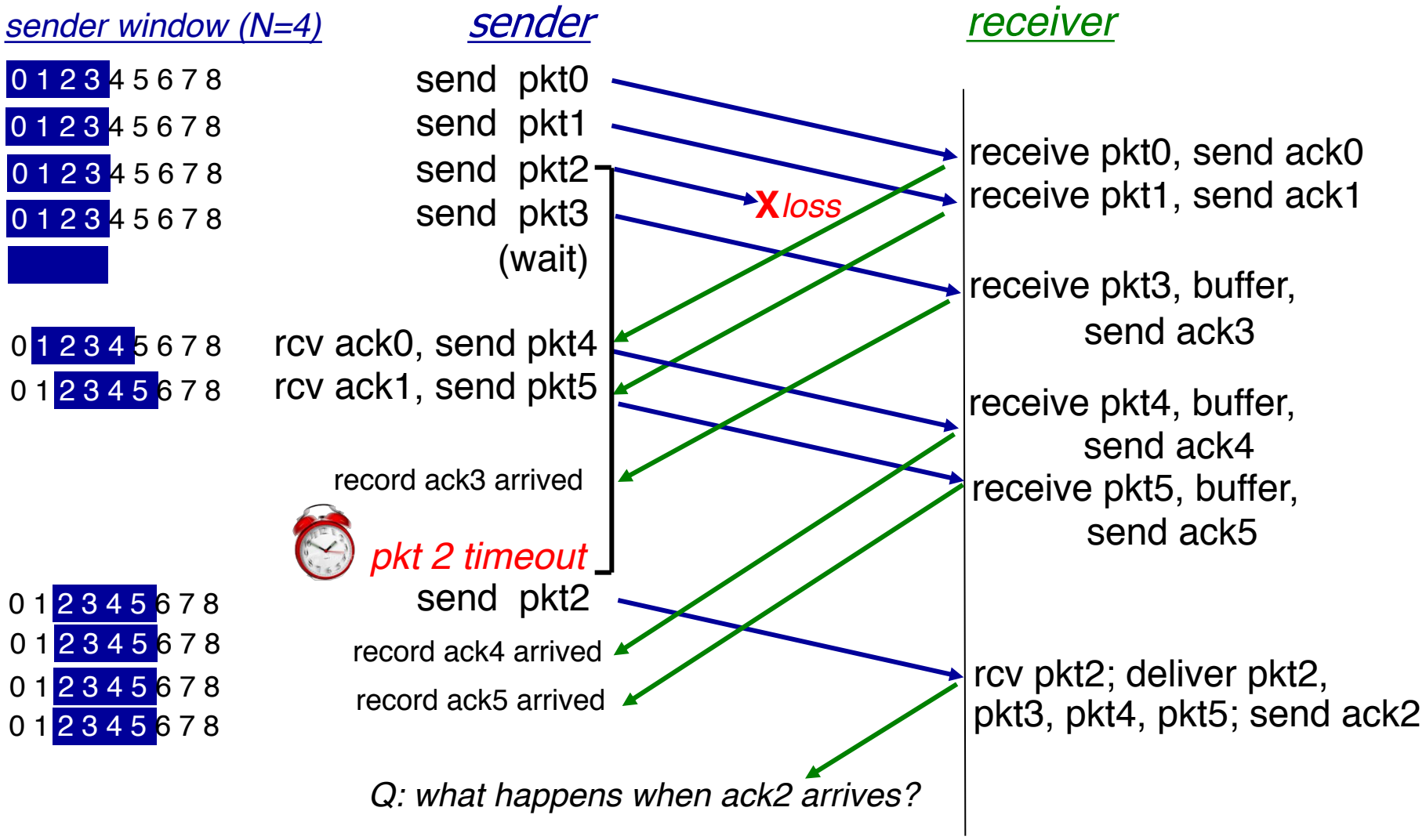
Go-back-N:

- 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 unacked packets in pipeline
- receiver sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet

Selective repeat in action

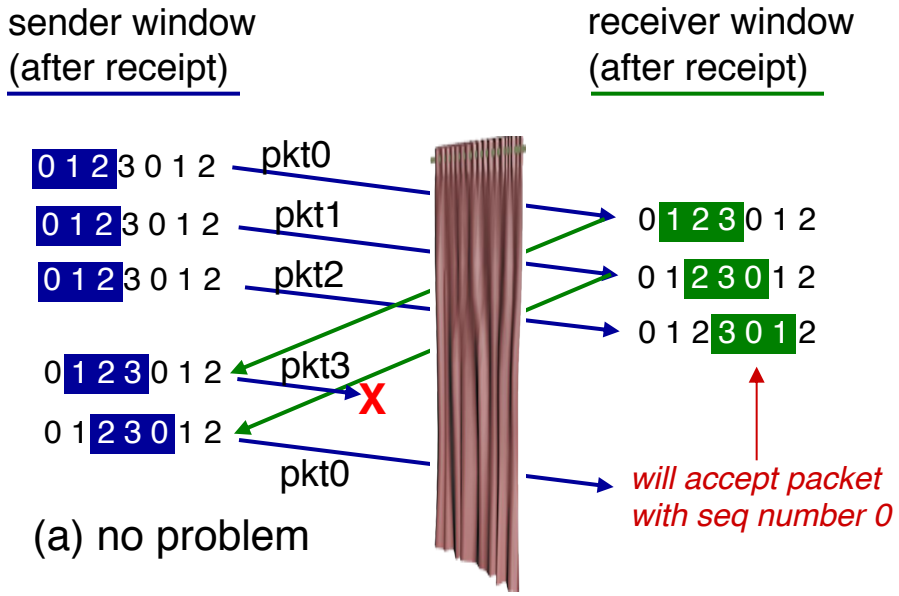


Selective repeat: dilemma

example:

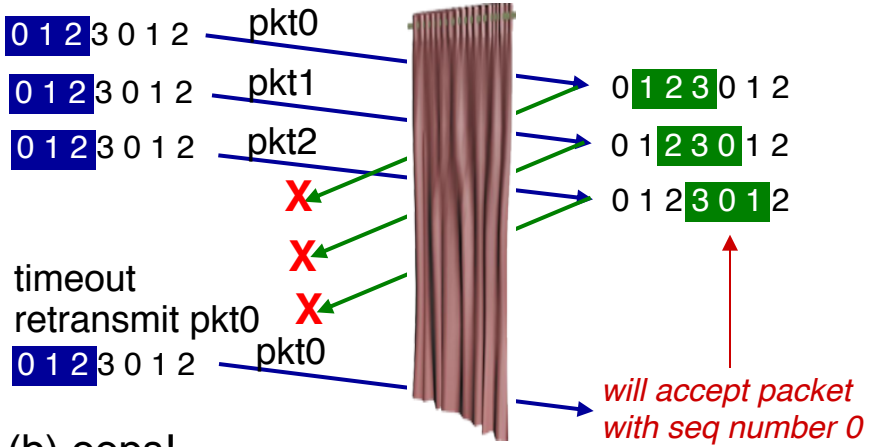
- seq #'s: 0, 1, 2, 3
- window size=3
  - receiver sees no difference in two scenarios!
  - duplicate data accepted as new in (b)

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!

Summary

- Reliability using sliding window/pipelined protocols
  - Continue to use concepts from stop and wait
  - ACKs, sequence numbers, timeouts

- Downside: More complexity, memory buffers on either side
- Benefit: higher throughput and resource utilization

- Sliding window protocol used extensively: TCP!