Transport Layer Protocols -2

A/PROF. DUY NGO

 COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

Learning Objectives

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

rdt2.0 Has a Fatal Flaw!

what happens if ACK / NAK corrupted?

- sender doesn't know what happened at receiver!
- Can't just retransmit: possible duplicate

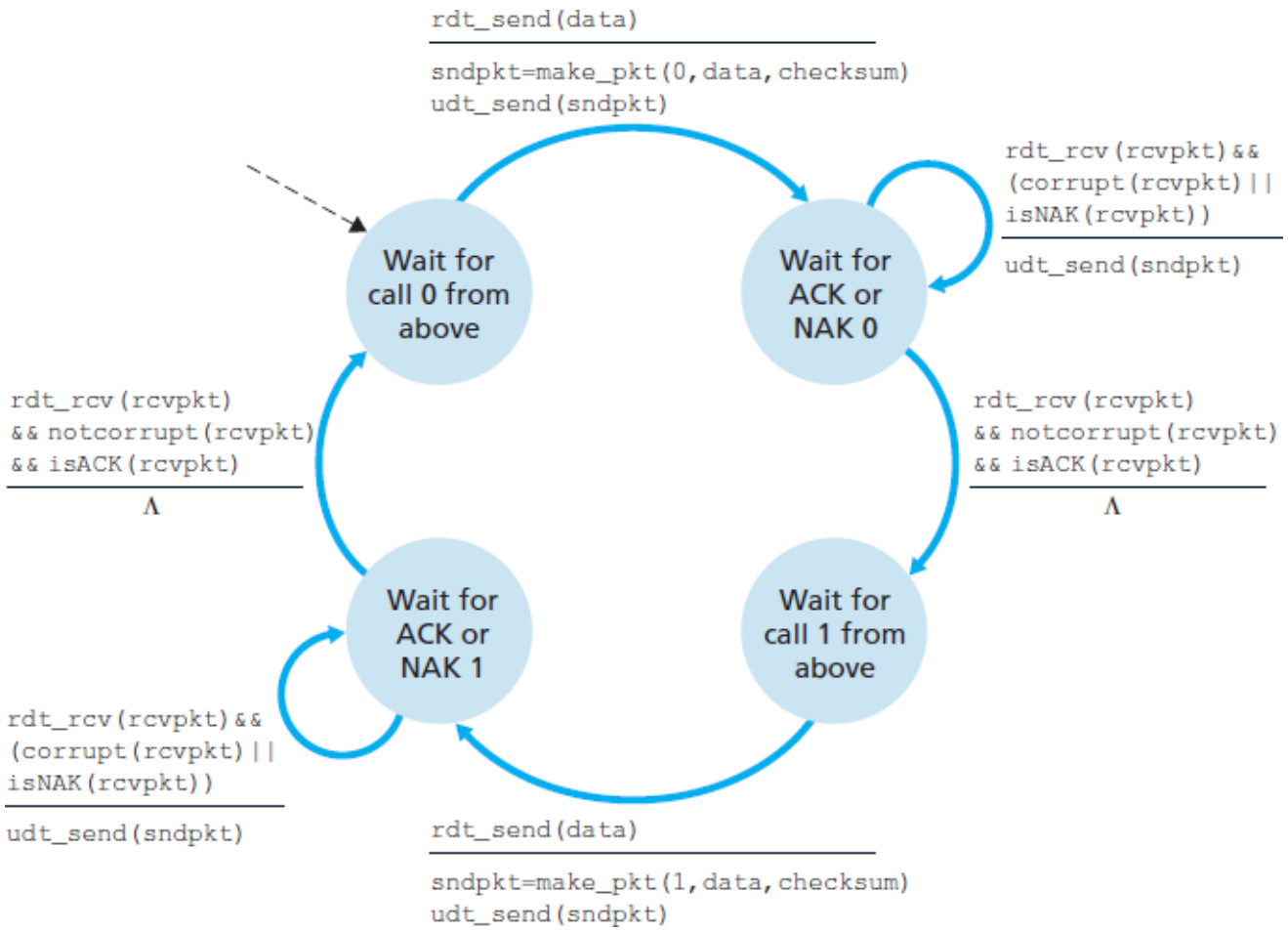
stop and wait

sender sends one packet, then waits for receiver response

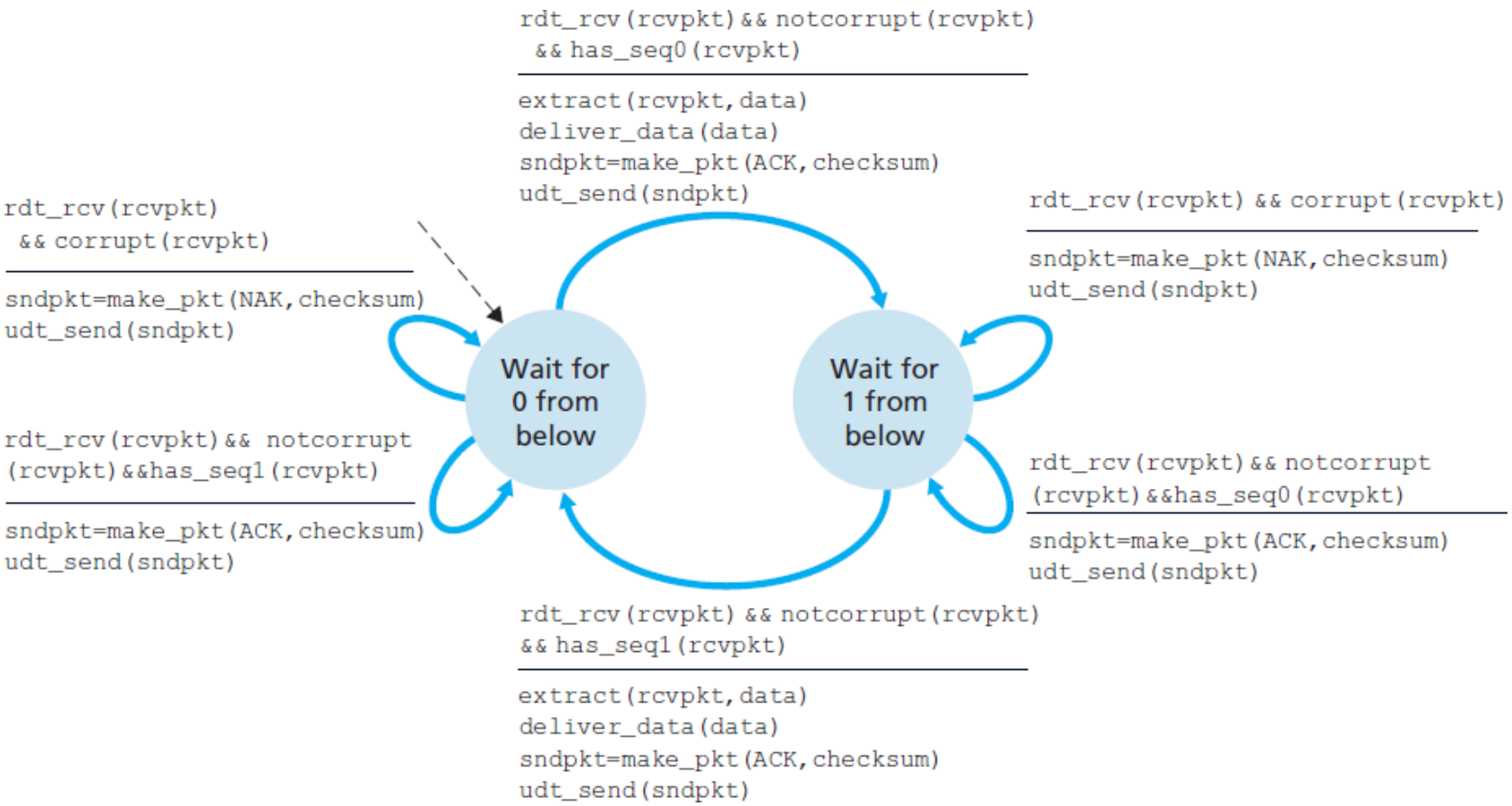
handling duplicates:

- sender retransmits current pkt if ACK / NAK corrupted
- sender adds **sequence number** to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

rdt2.1: Sender, Handles *Garbled* ACK/NAK s



rdt2.1: Receiver, Handles Garbled ACK/NAKs



rdt2.1: Discussion

sender:

- seq # added to pkt
- two Sequence #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "expected" pkt should have seq # of 0 or 1

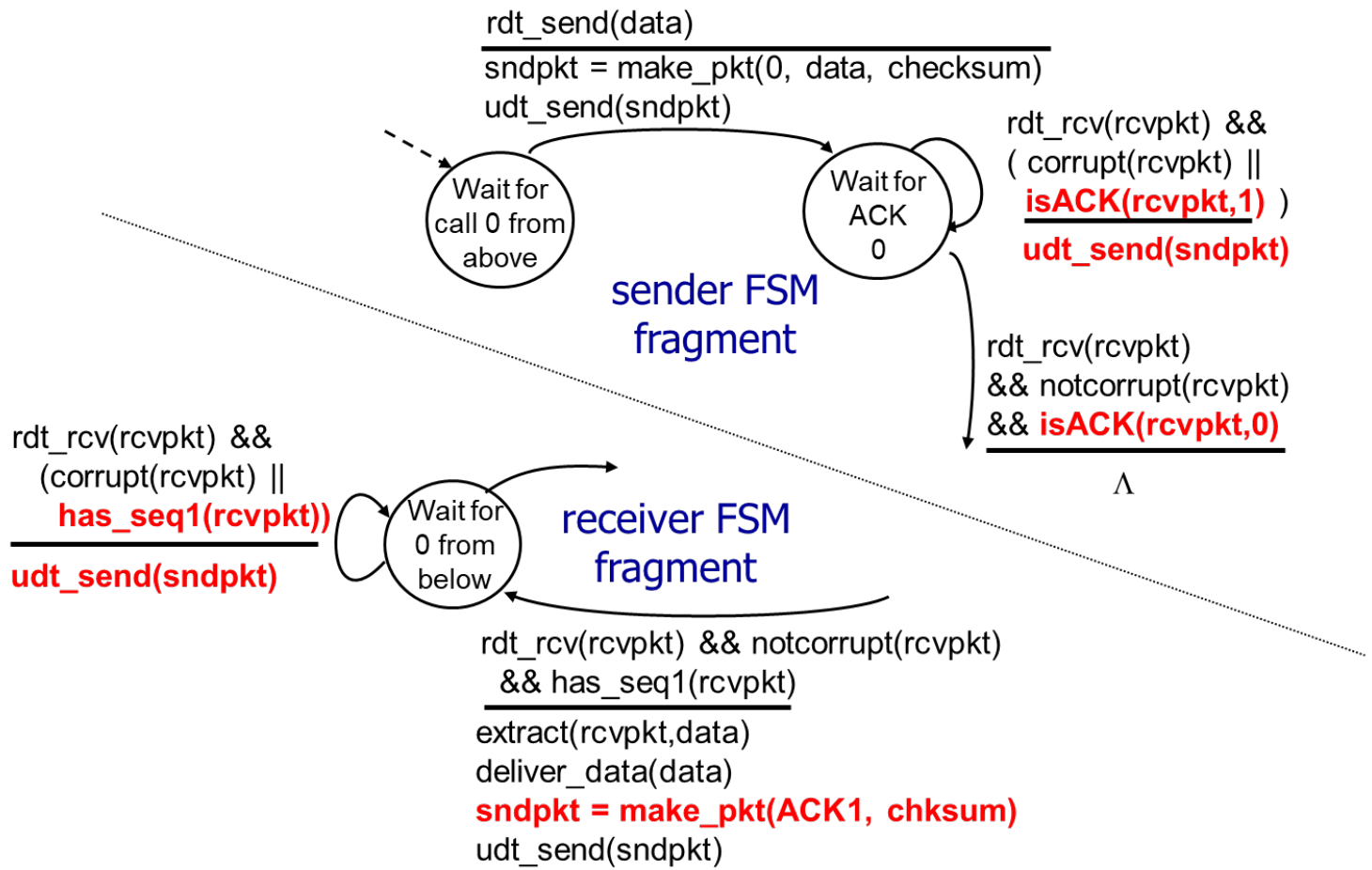
receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can **not** know if its last ACK/NAK received OK at sender

rdt2.2: A NAK - free Protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must **explicitly** include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: **retransmit current pkt**

rdt2.2: Sender, Receiver Fragments



rdt3.0: Channels with Errors and Loss

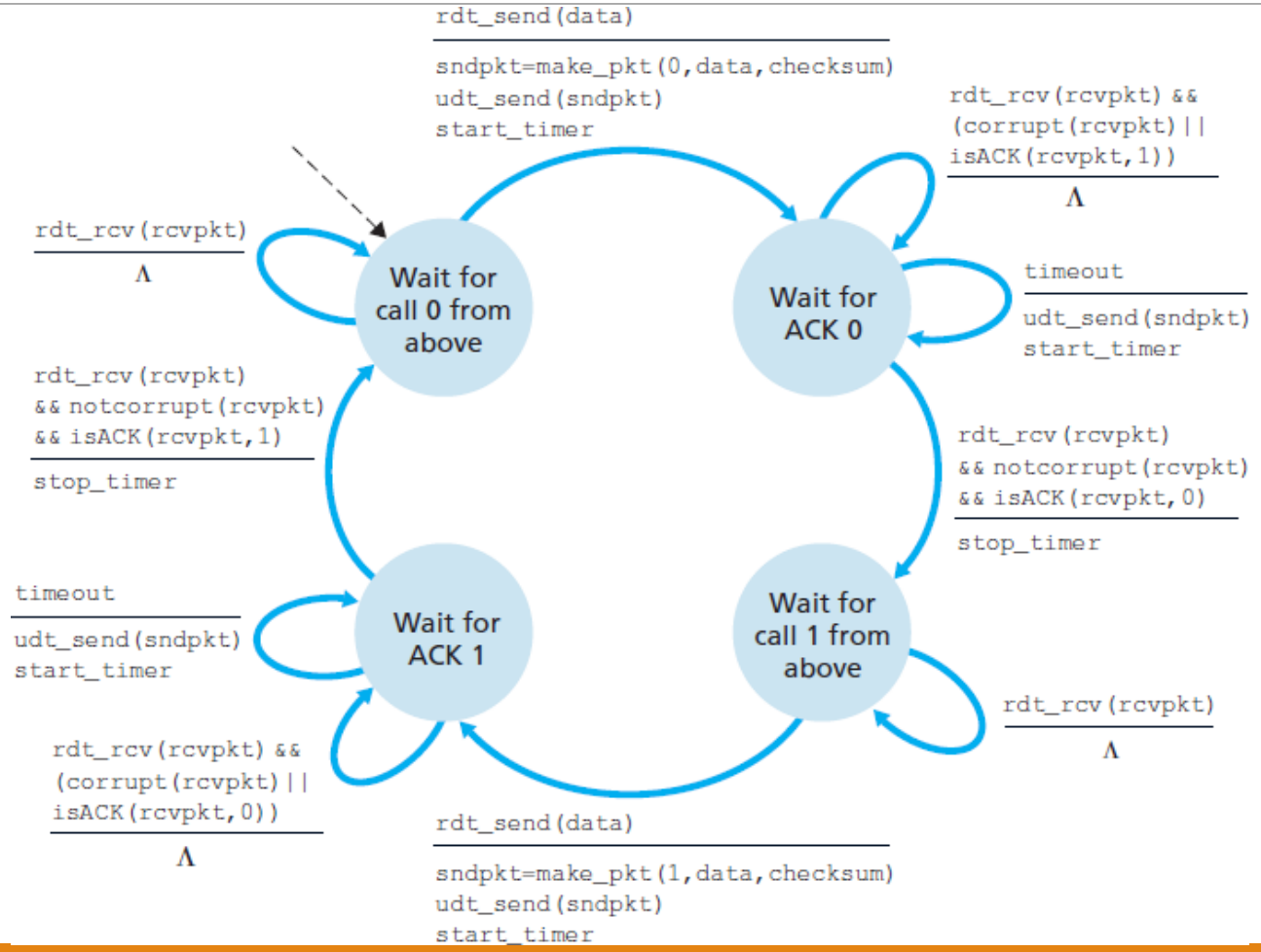
new assumption: underlying channel can also lose packets (data, ACKs)

- checksum, Sequence #, ACKs, retransmissions will be of help … but not enough

approach: sender waits "reasonable" amount of time for ACK

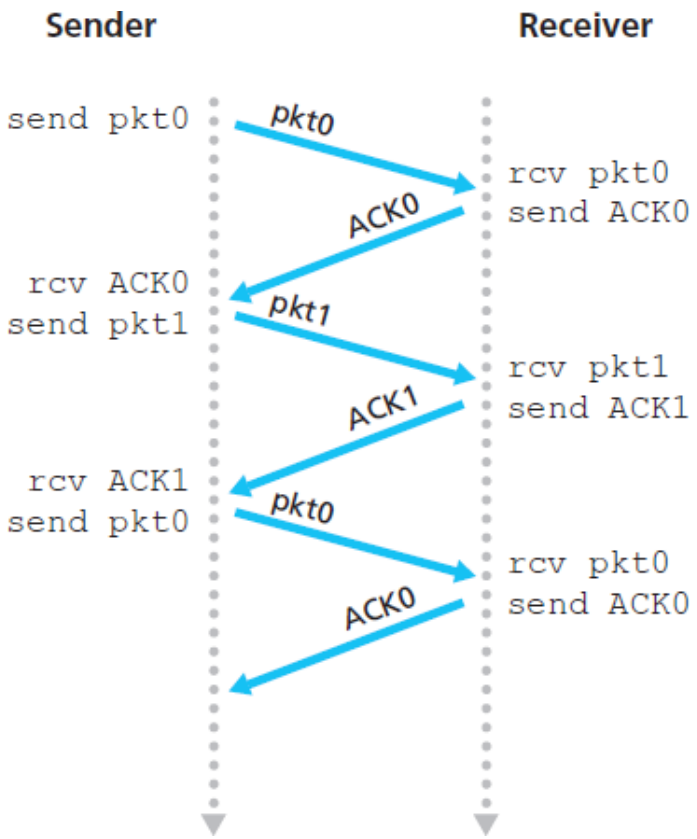
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but Sequence #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

rdt3.0 Sender

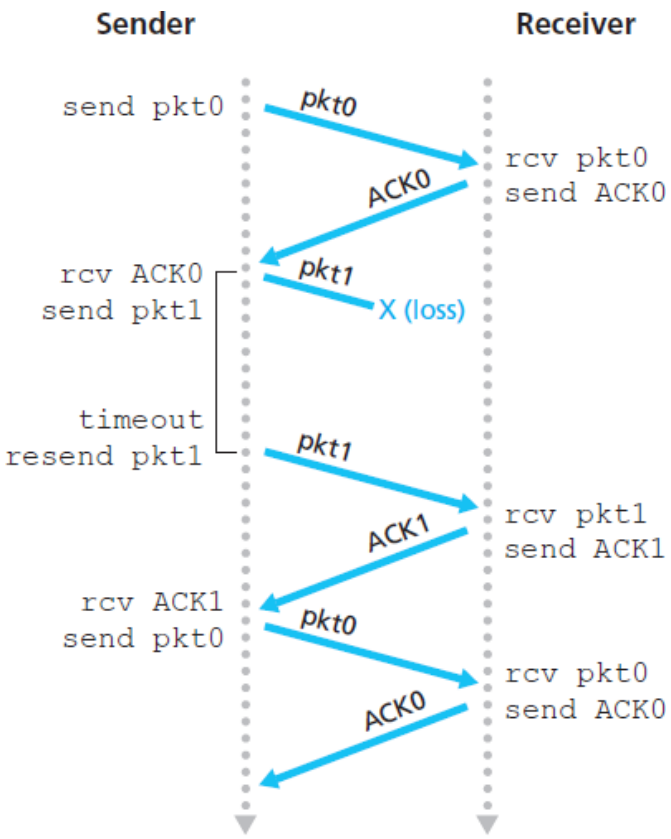


rdt3.0 in Action (1 of 2)

(a) no loss

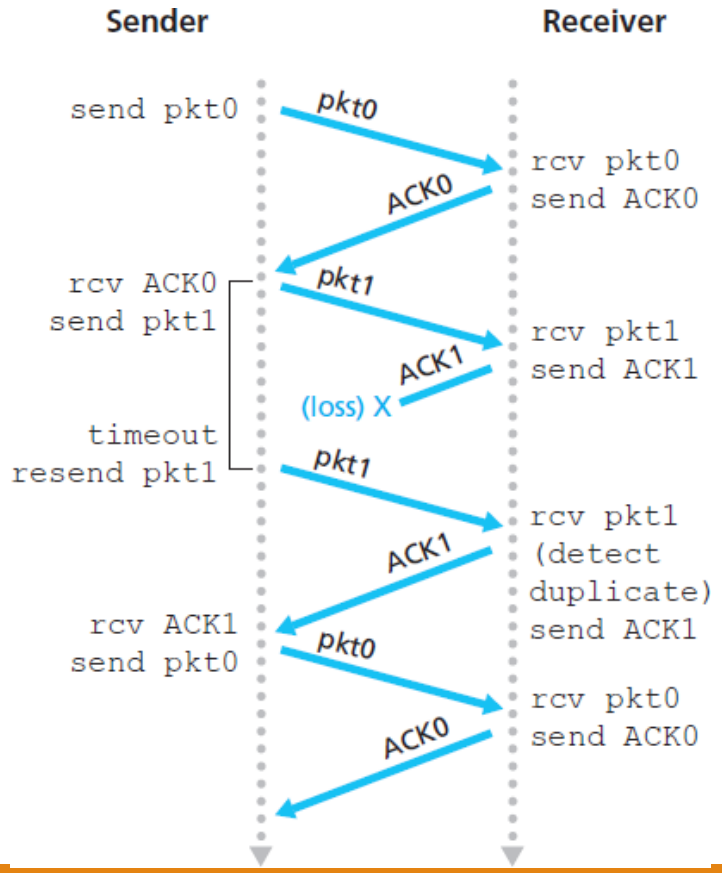


(b) packet loss

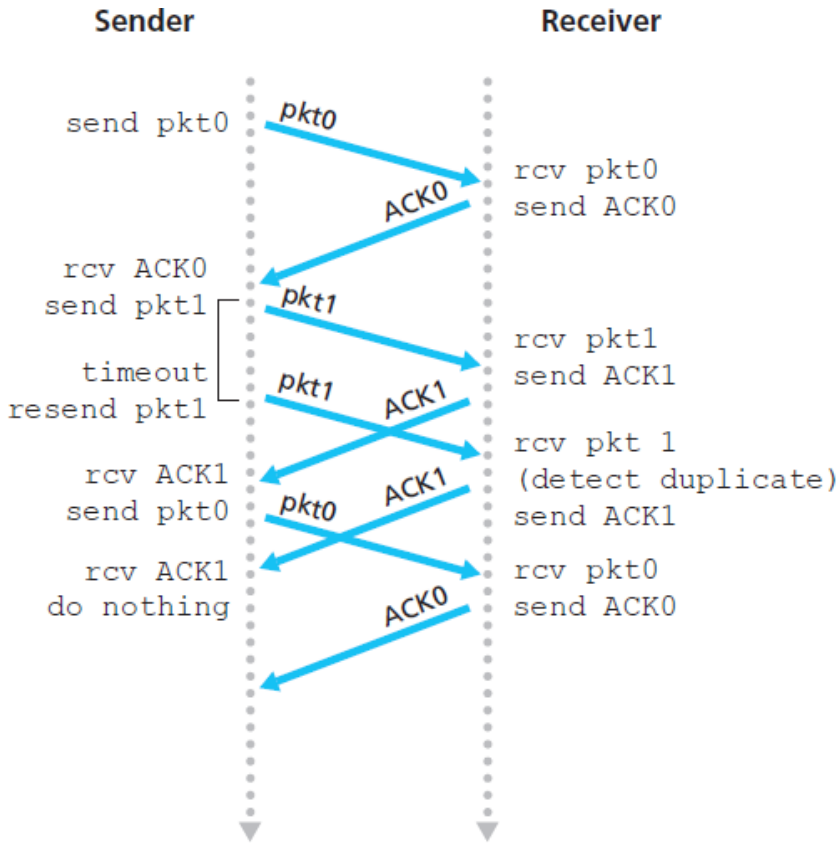


rdt3.0 in Action (2 of 2)

(c) ACK loss



(d) premature timeout/ delayed ACK



Performance of rdt3.0 (1 of 2)

- rdt3.0 is correct, but performance not so good
- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

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

Performance of rdt3.0 (2 of 2)

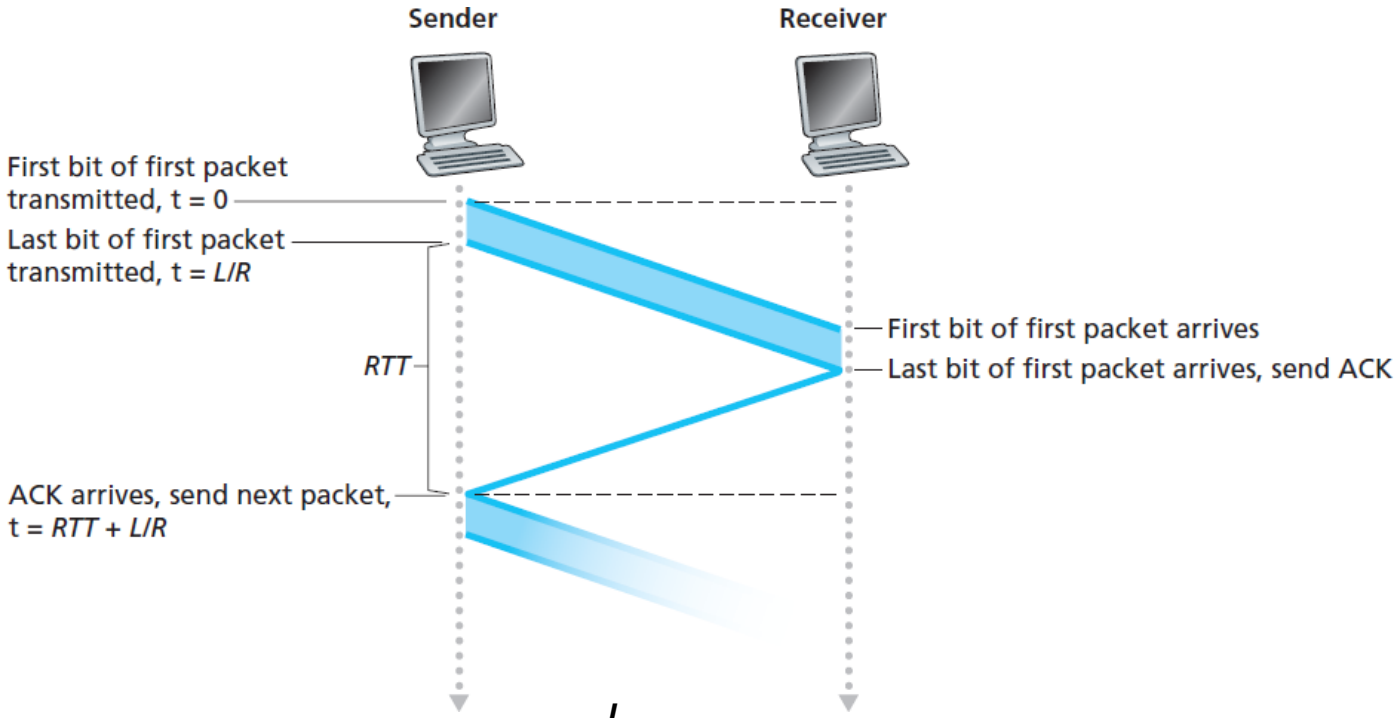
- U_{sender}: **utilization** – fraction of time sender busy sending

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

  - if RTT = 30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link

- network protocol limits use of physical resources!

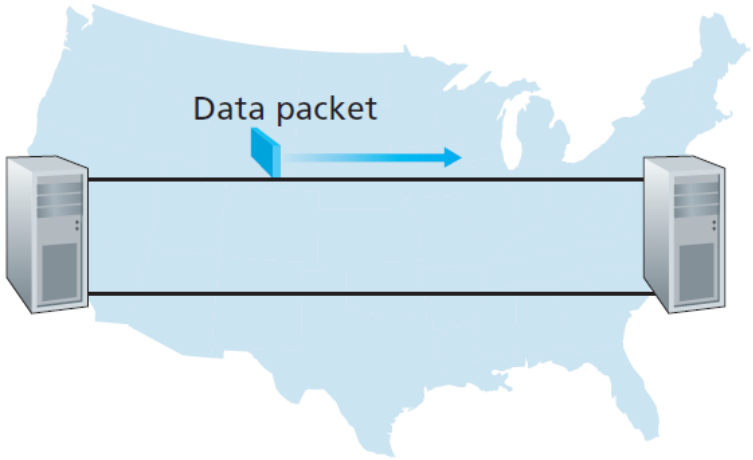
rdt3.0: Stop-and-wait Operation



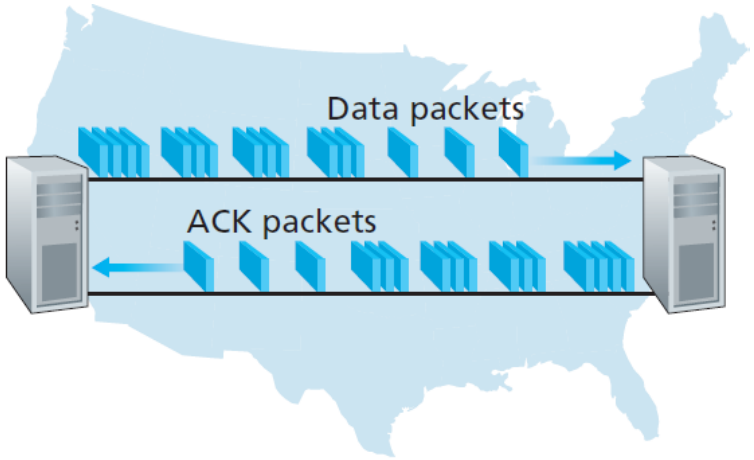
$$U_{sender} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{.008}{30.008} = 0.00027$$

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



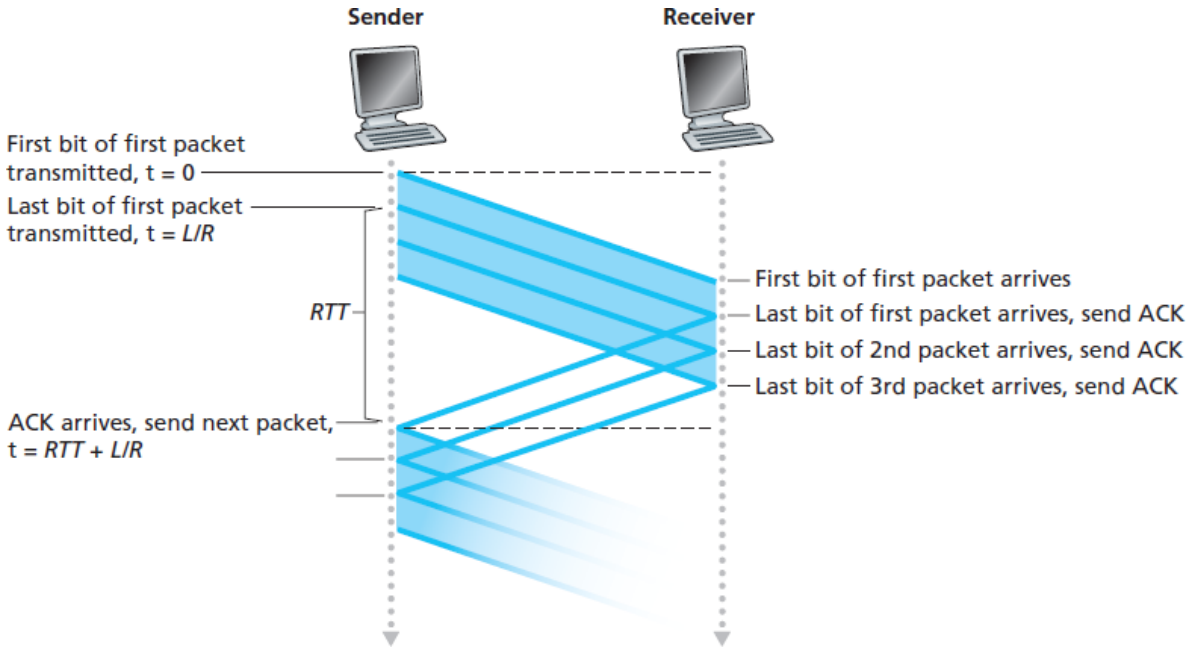
a. A stop-and-wait protocol in operation



b. A pipelined protocol in operation

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

Pipelining: Increased Utilization



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

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

Pipelined Protocols: Overview

Go-back-N:

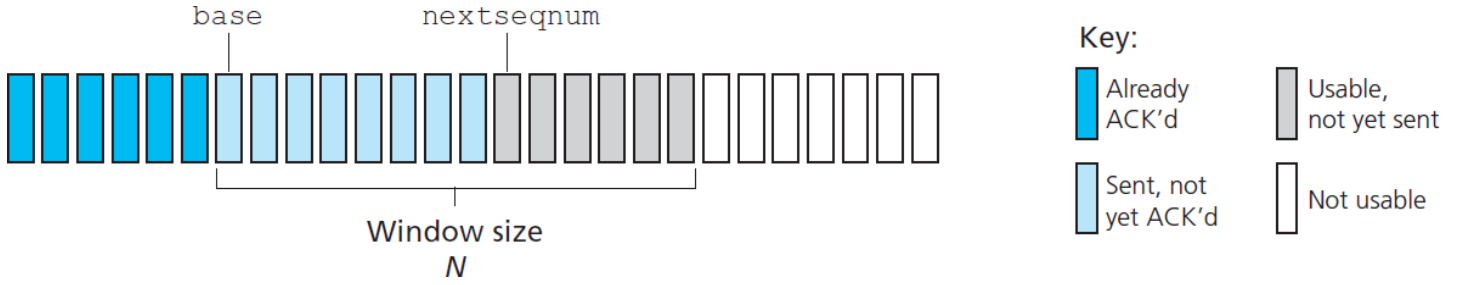
- sender can have up to N unacked packets in pipeline
- receiver only sends **cumulative ack**
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit **all** unacked packets

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

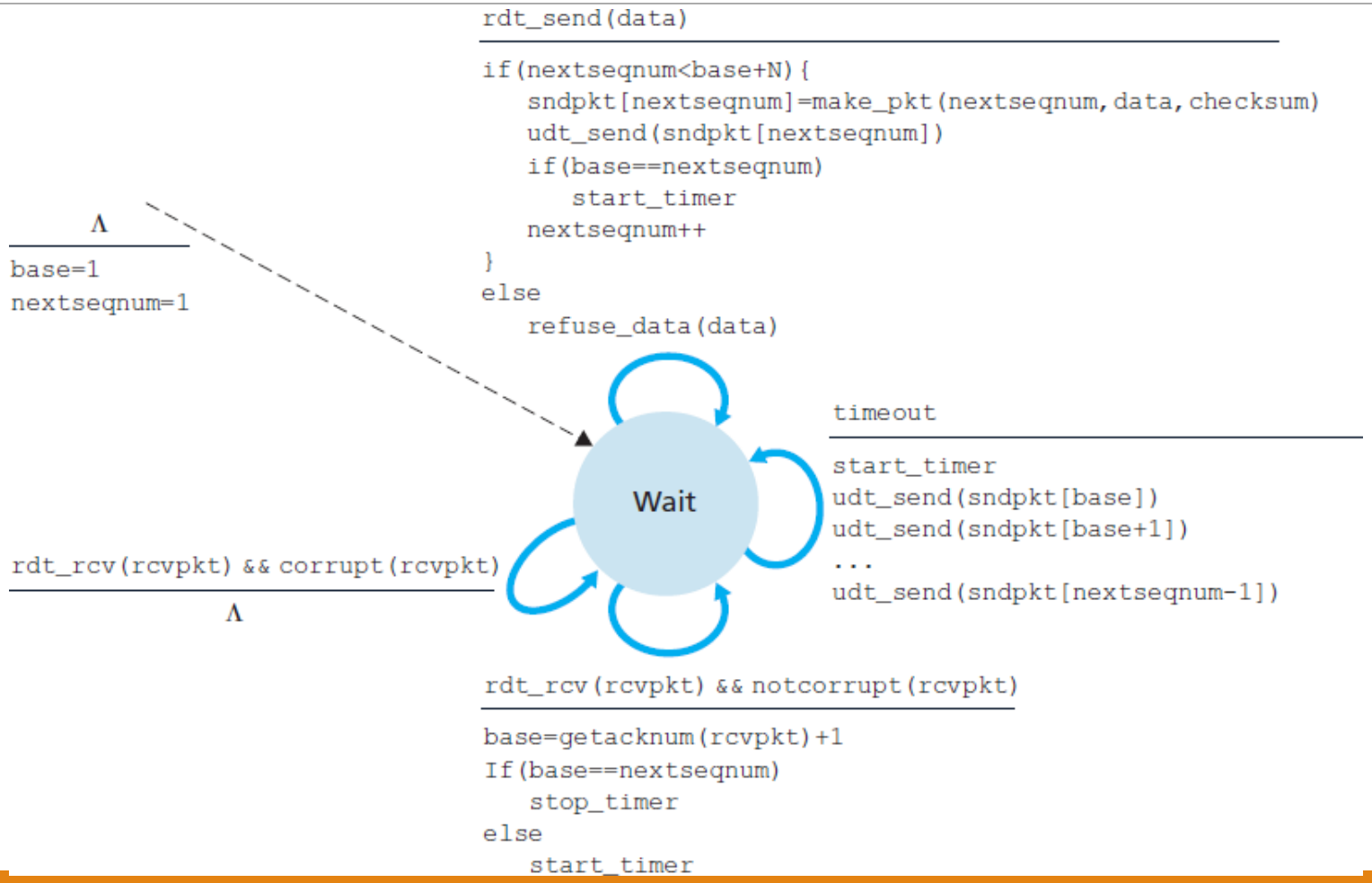
Go-Back-N: Sender

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

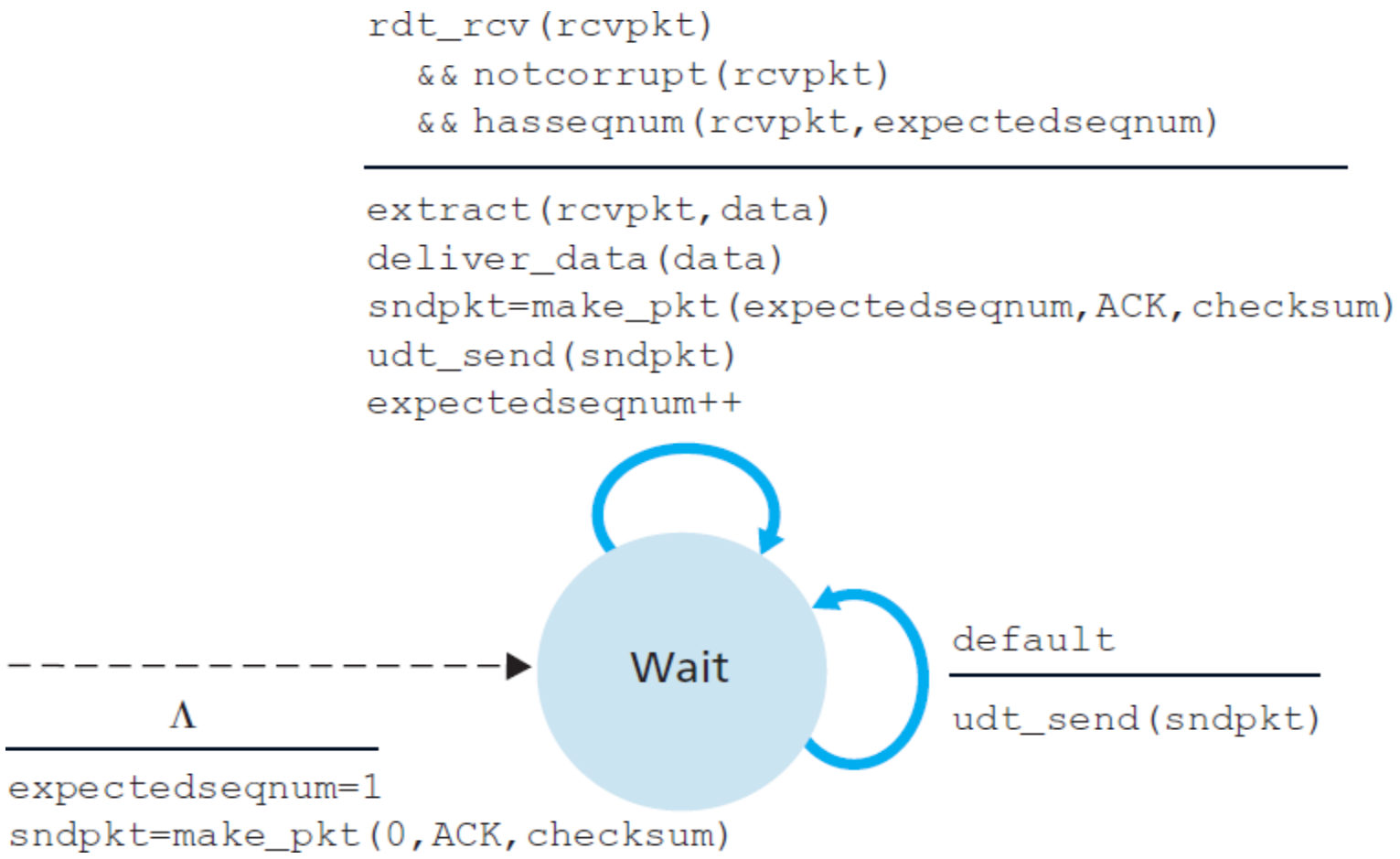


- ACK (n): ACKs all pkts up to, including seq # n – **"cumulative ACK"**
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- **timeout(n):** retransmit packet n and all higher seq # pkts in window

GBN: Sender Extended FSM



GBN: Receiver Extended FSM (1 of 2)



GBN: Receiver Extended FSM (2 of 2)

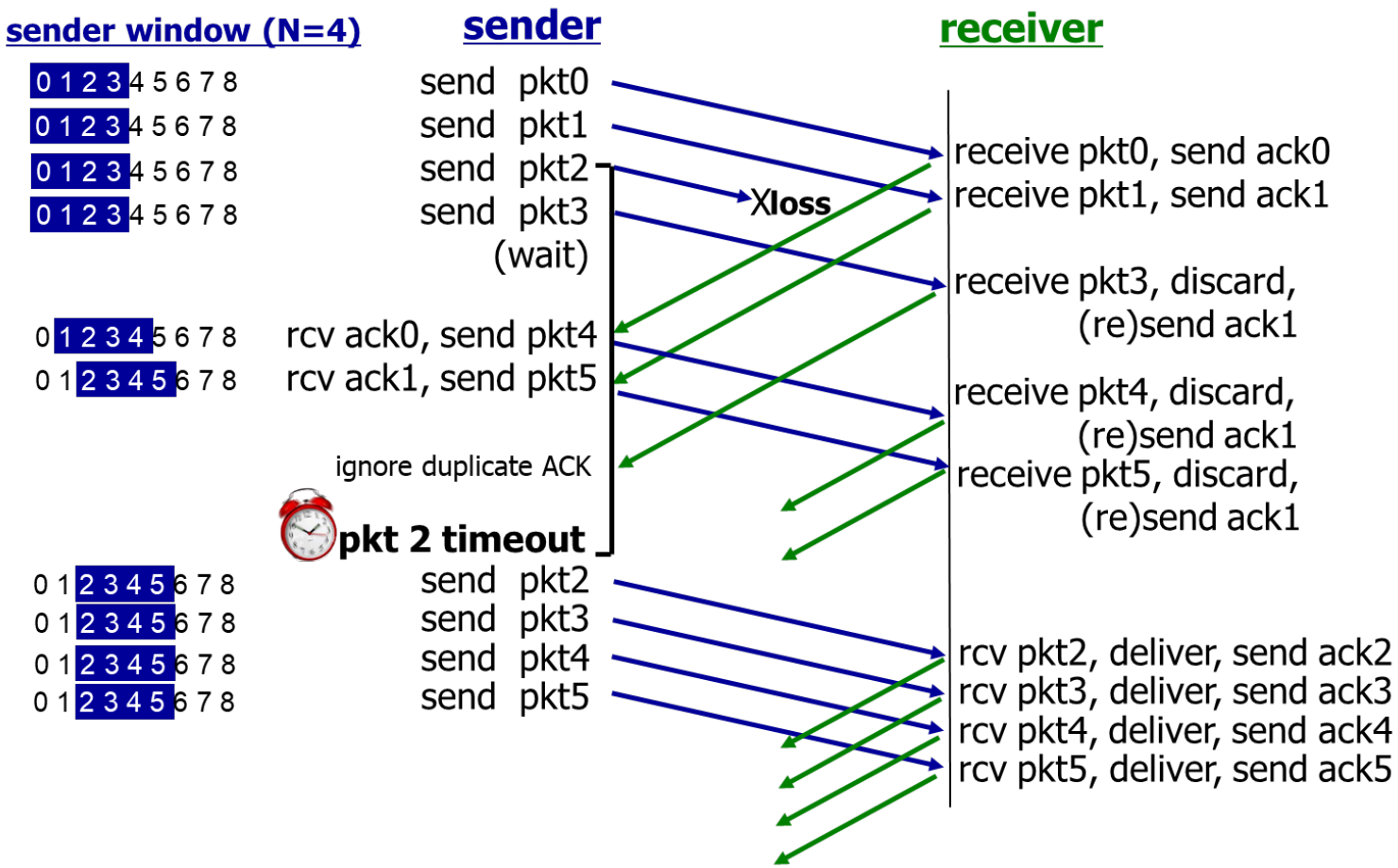
ACK-only: always send ACK for correctly-received pkt with highest **in-order** seq #

- 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 #

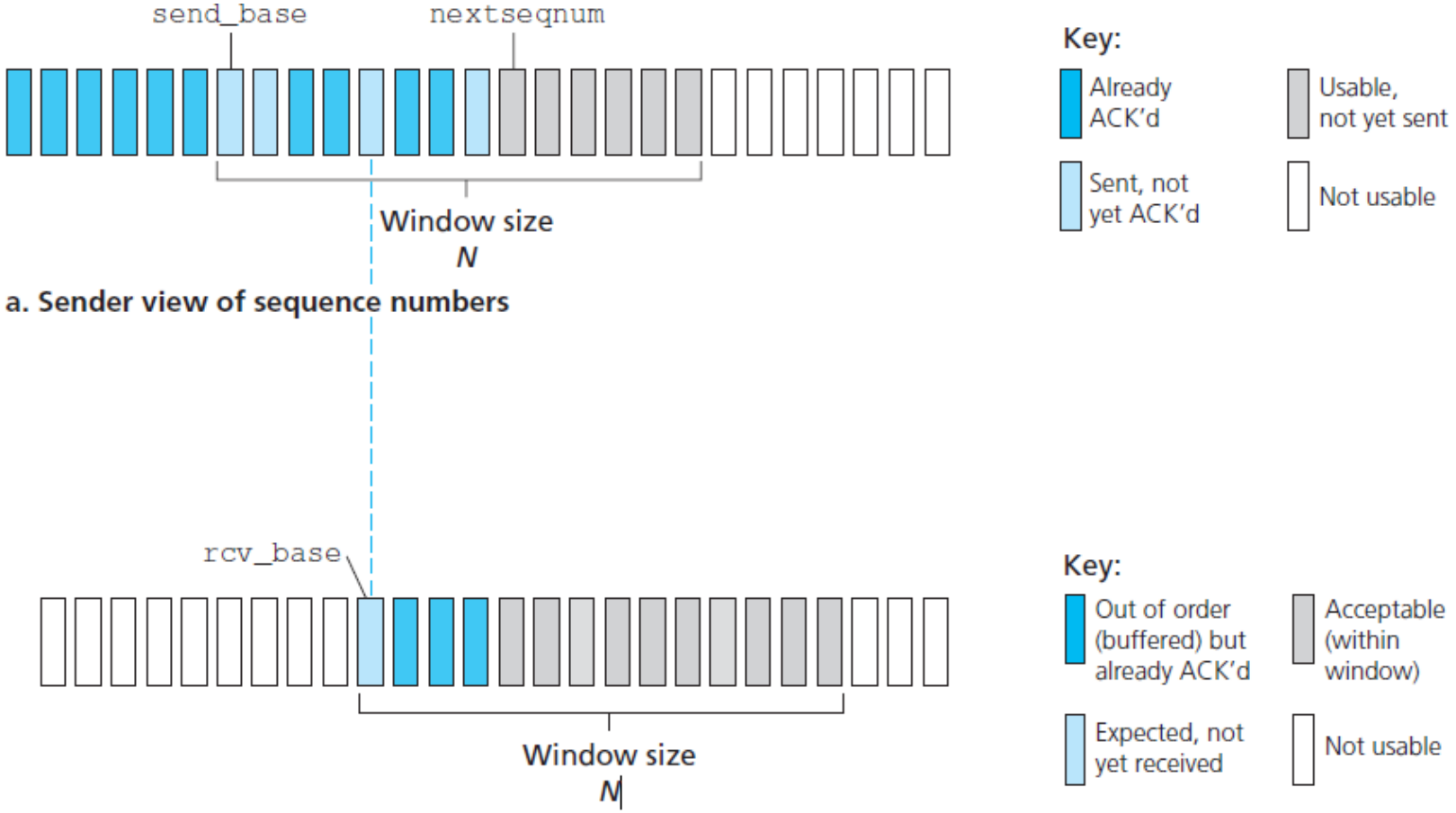
GBN in Action



Selective Repeat (1 of 3)

- 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 (2 of 3)

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 #

Selective Repeat (3 of 3)

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

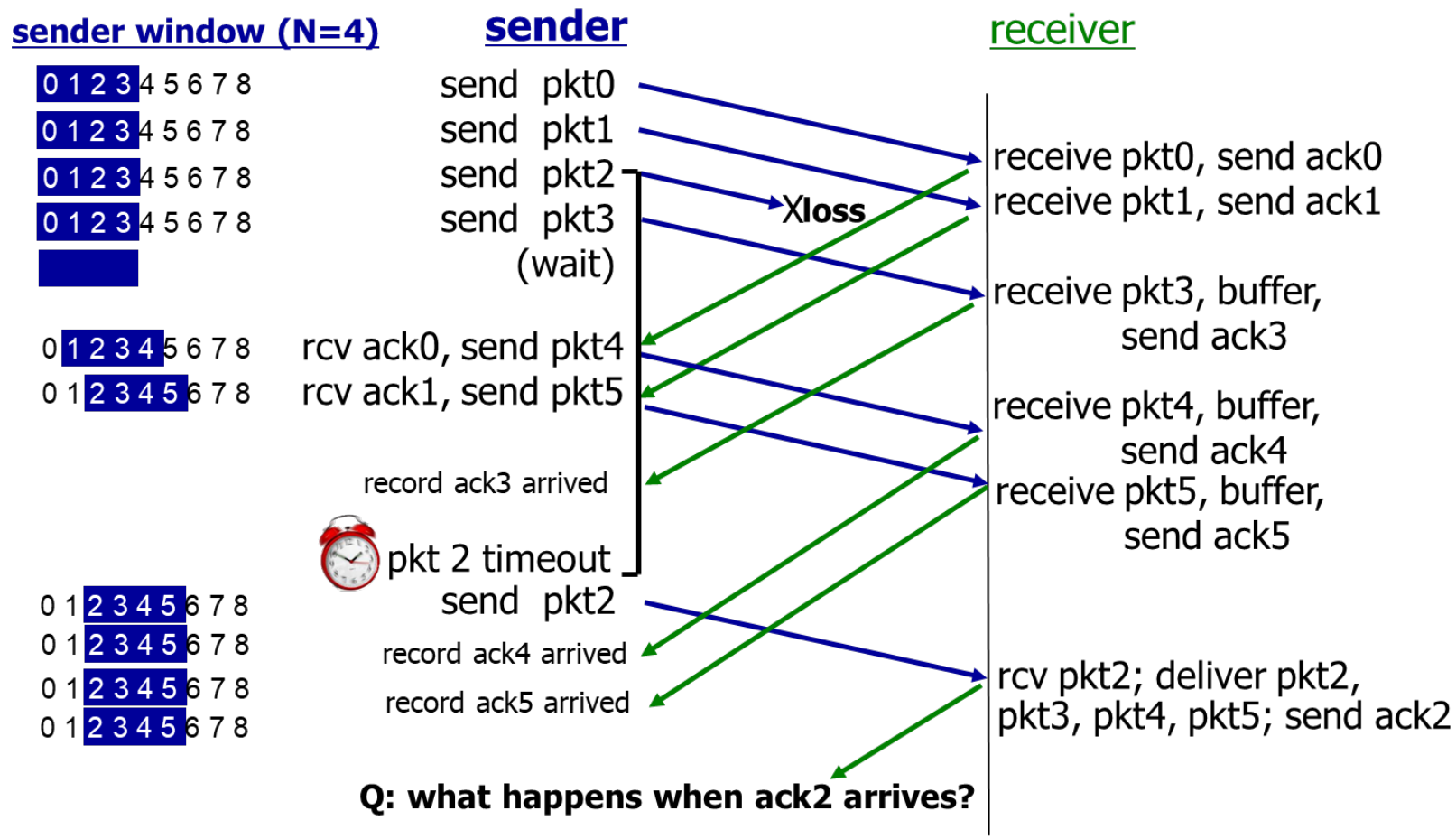
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

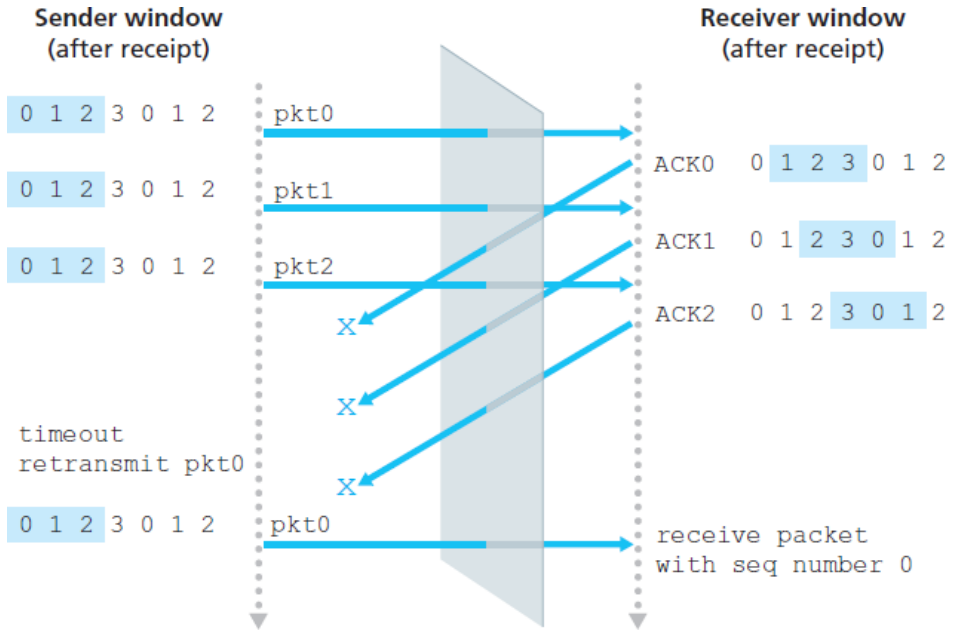
Selective Repeat in Action



Selective Repeat: Dilemma (1 of 2)

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

(a) no problem



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

Selective Repeat: Dilemma (2 of 2)

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

(b) oops!