National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 12

Chapter 3

4th October, 2022

Nauman Moazzam Hayat

nauman.moazzam@lhr.nu.edu.pk

Office Hours: 02:30 pm till 06:00 pm (Every Tuesday & Thursday)

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

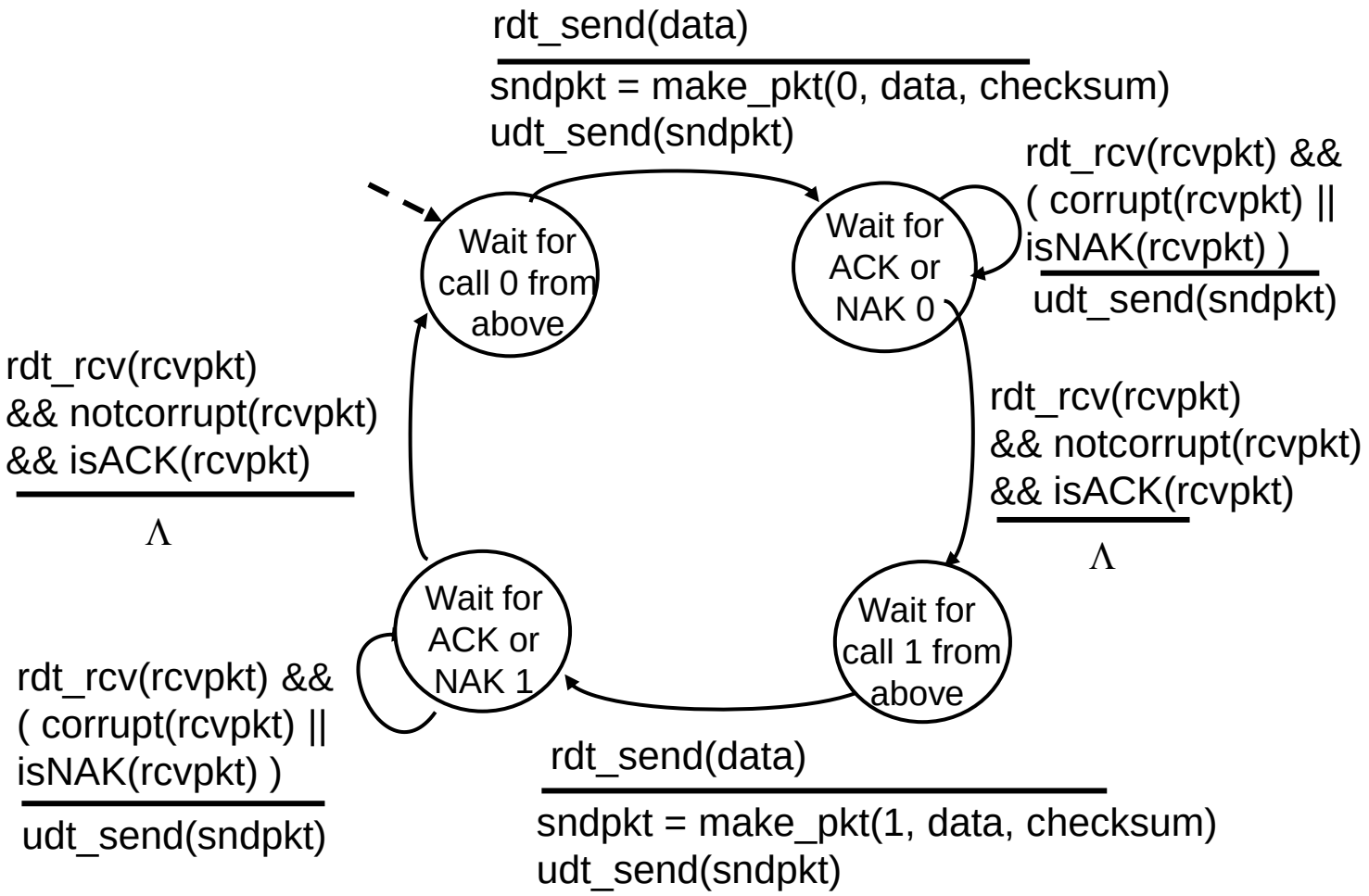
handling duplicates:

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

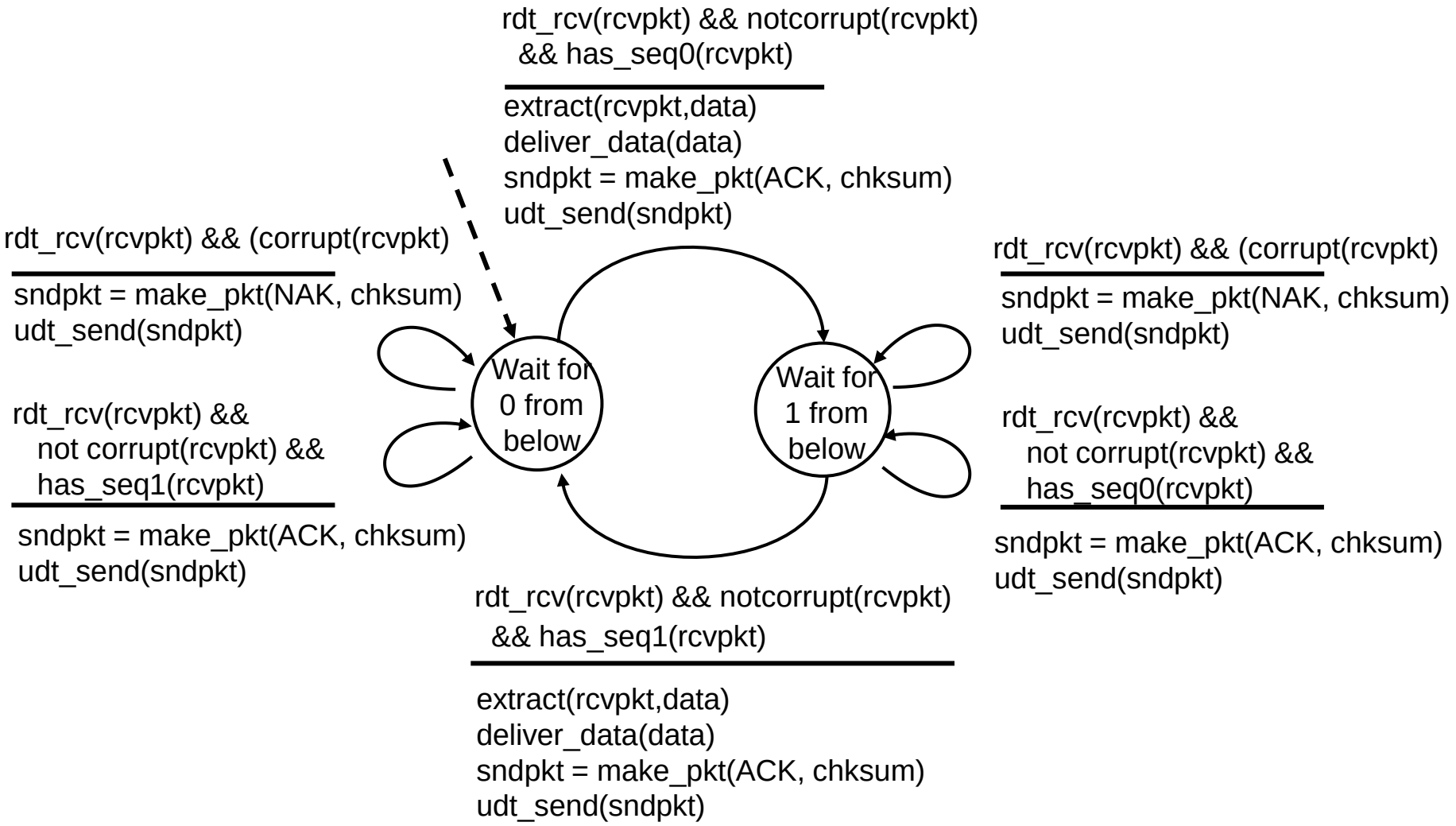
stop and wait

sender sends one packet, then waits for receiver response

rdt2.1: sender, handles garbled ACK/NAKs



rdt2.1: receiver, handles garbled ACK/NAKs



rdt2.1: discussion

sender:

- seq # added to pkt
- two seq. #'s (0,1) will suffice. Why? (since it is a simple stop and wait protocol, if receiver receives the same sequence number twice, it knows it is duplicate.)
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "expected" pkt should have seq # of 0 or 1

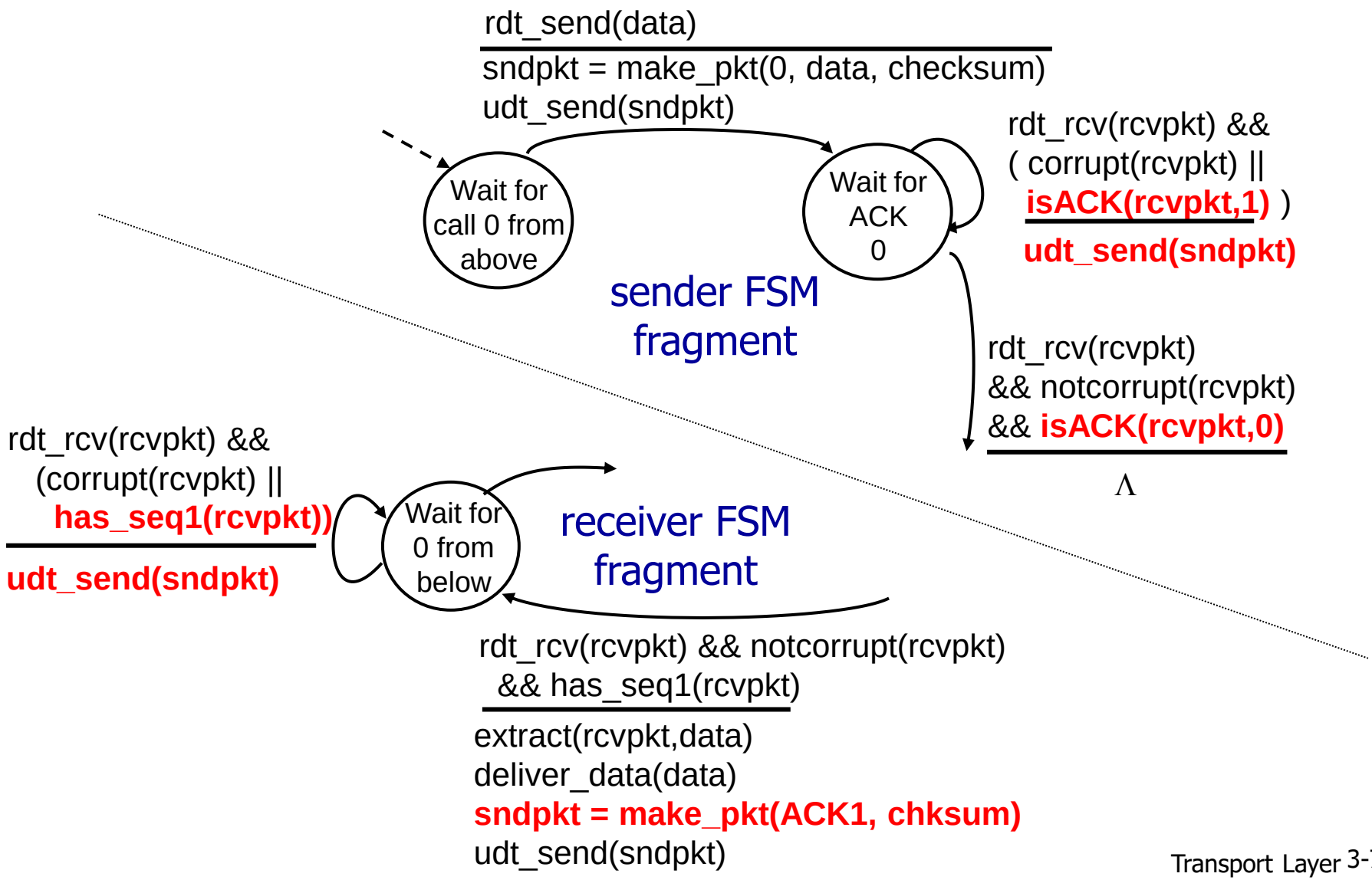
receiver:

- 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:

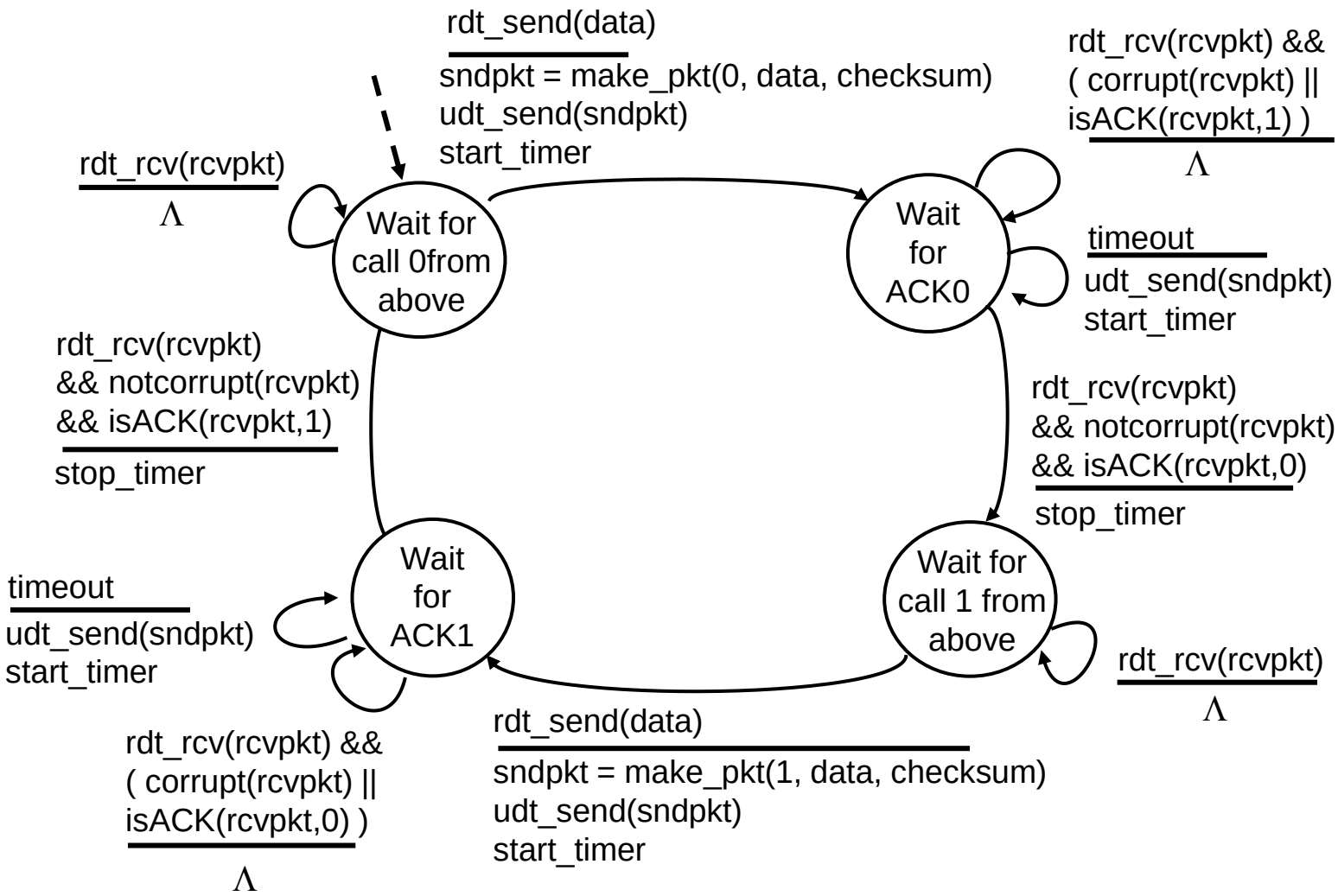
underlying channel can also lose packets (data, ACKs)

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

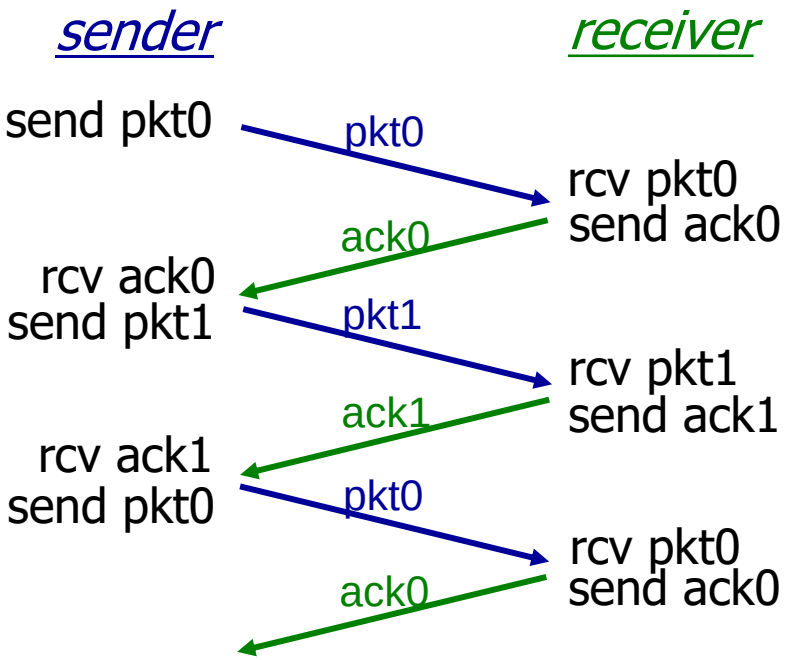
approach: sender waits "reasonable" amount of time for ACK

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

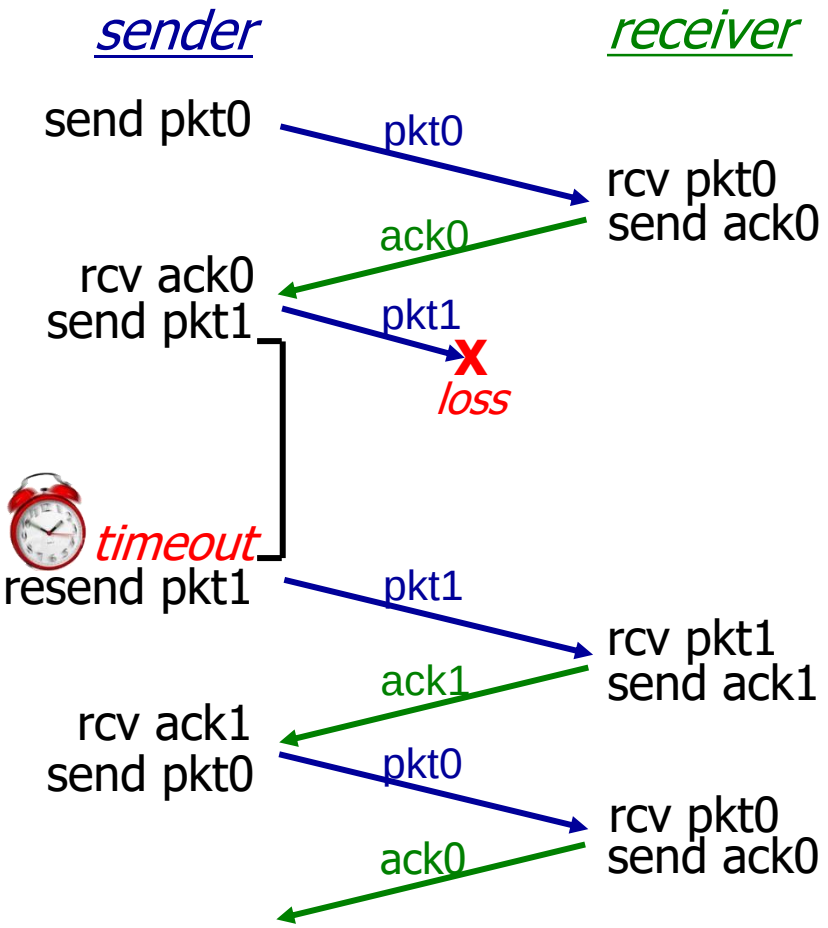
rdt3.0 sender



rdt3.0 in action

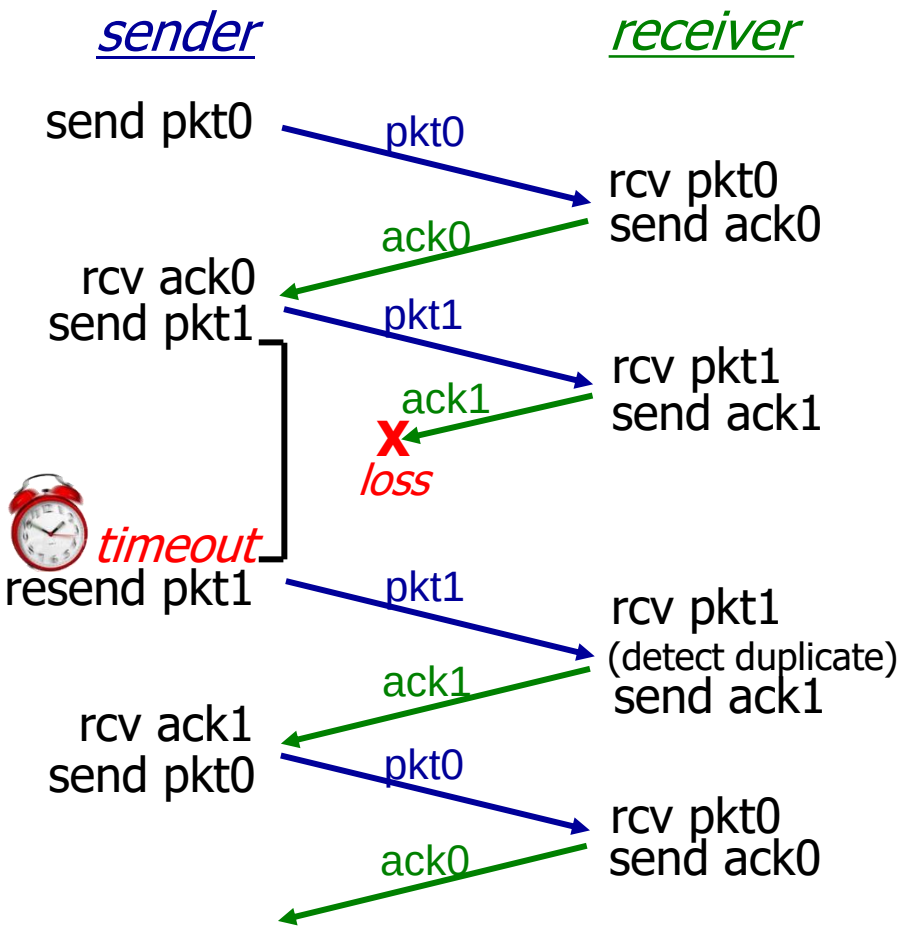


(a) no loss

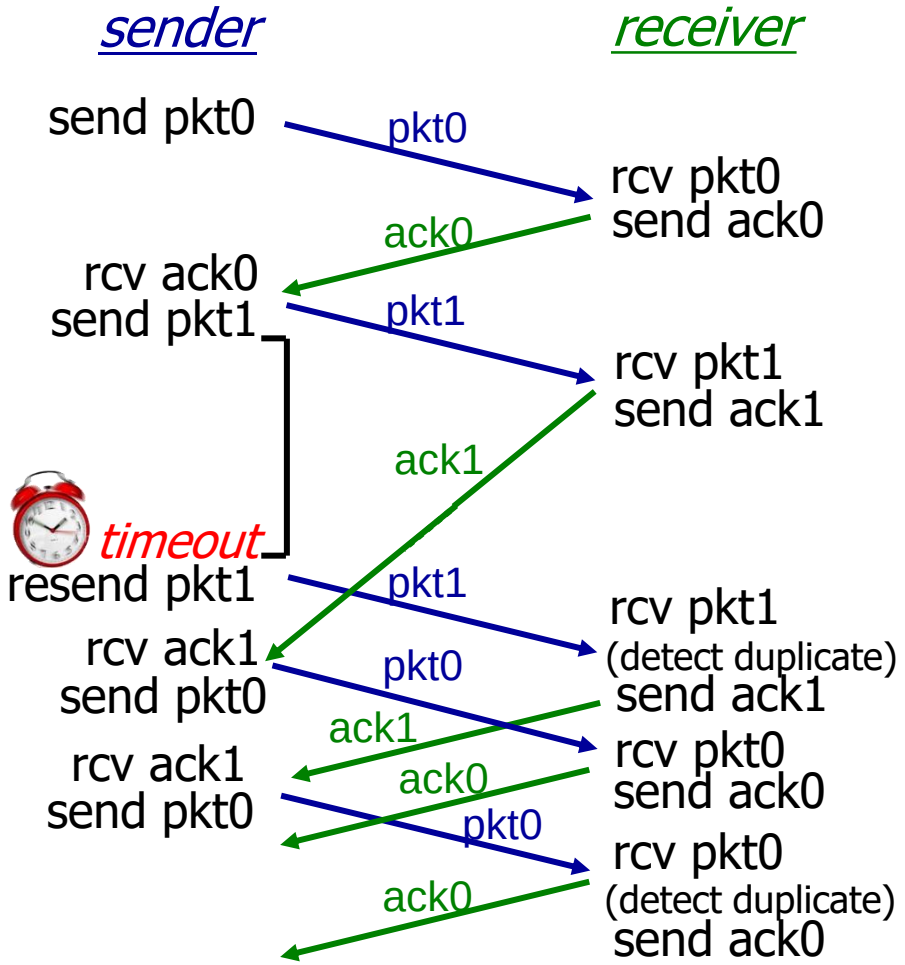


(b) packet loss

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

Performance of rdt3.0

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

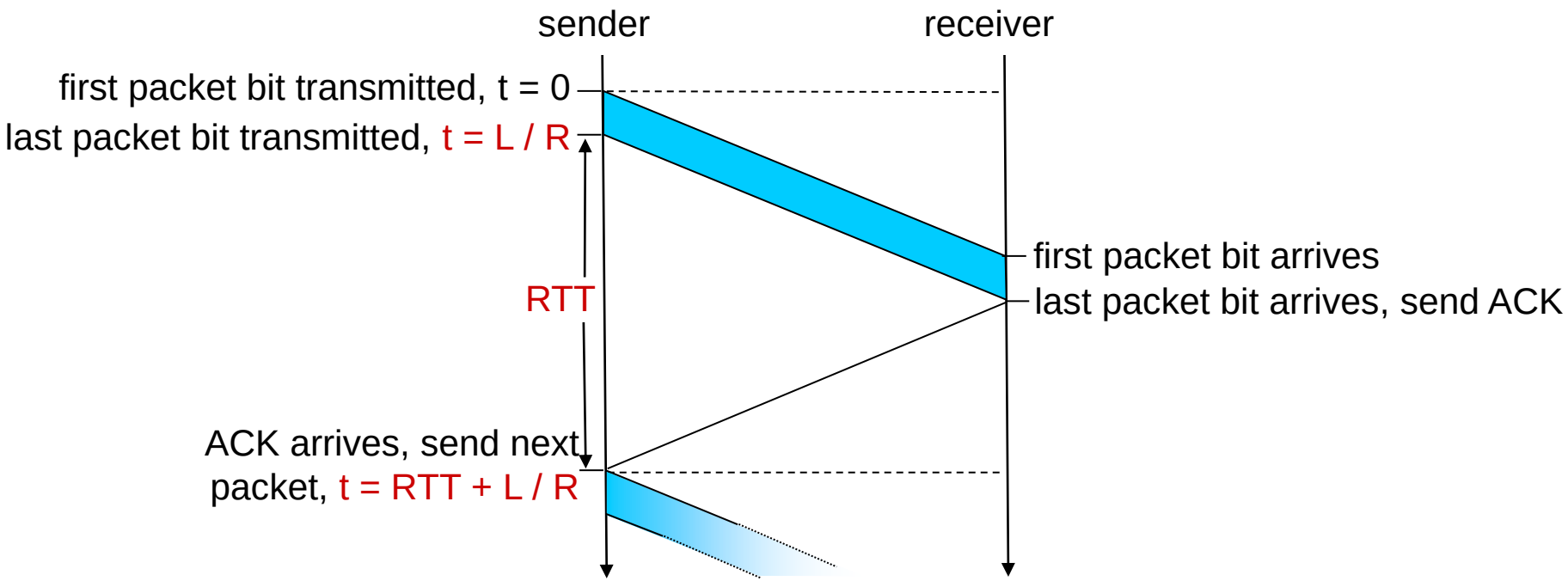
$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

  - 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

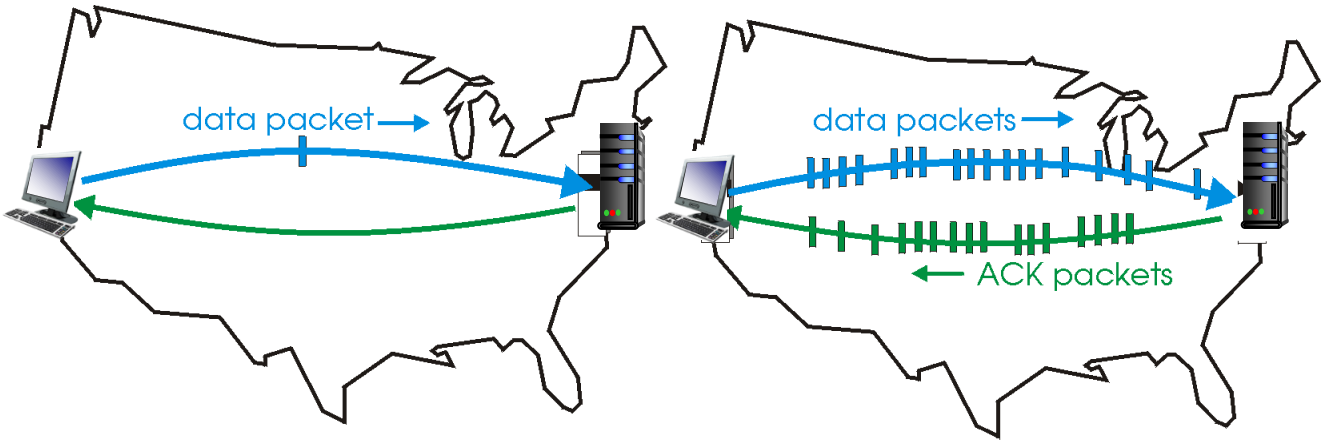


$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

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

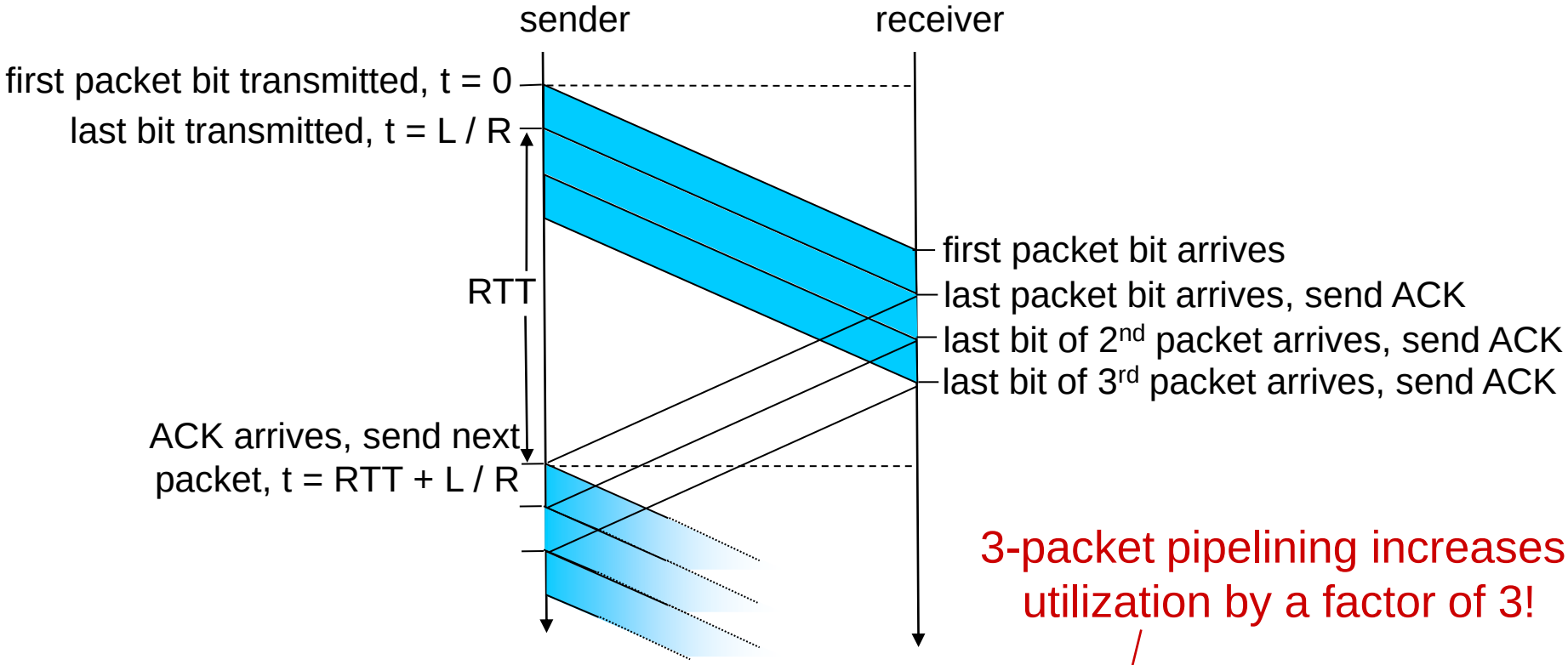


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- two generic forms of pipelined protocols: *go-Back-N (also called sliding window protocol) & 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: (Sliding Window)

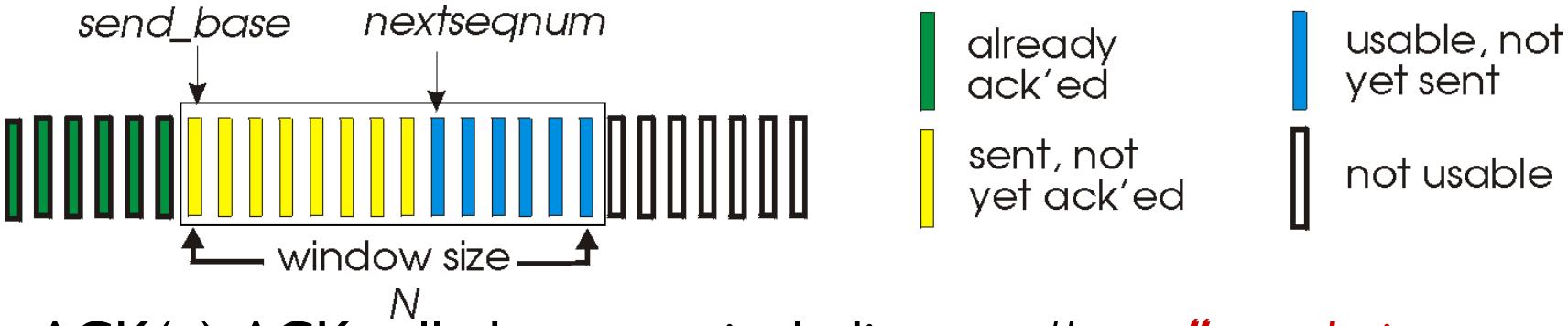
- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

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

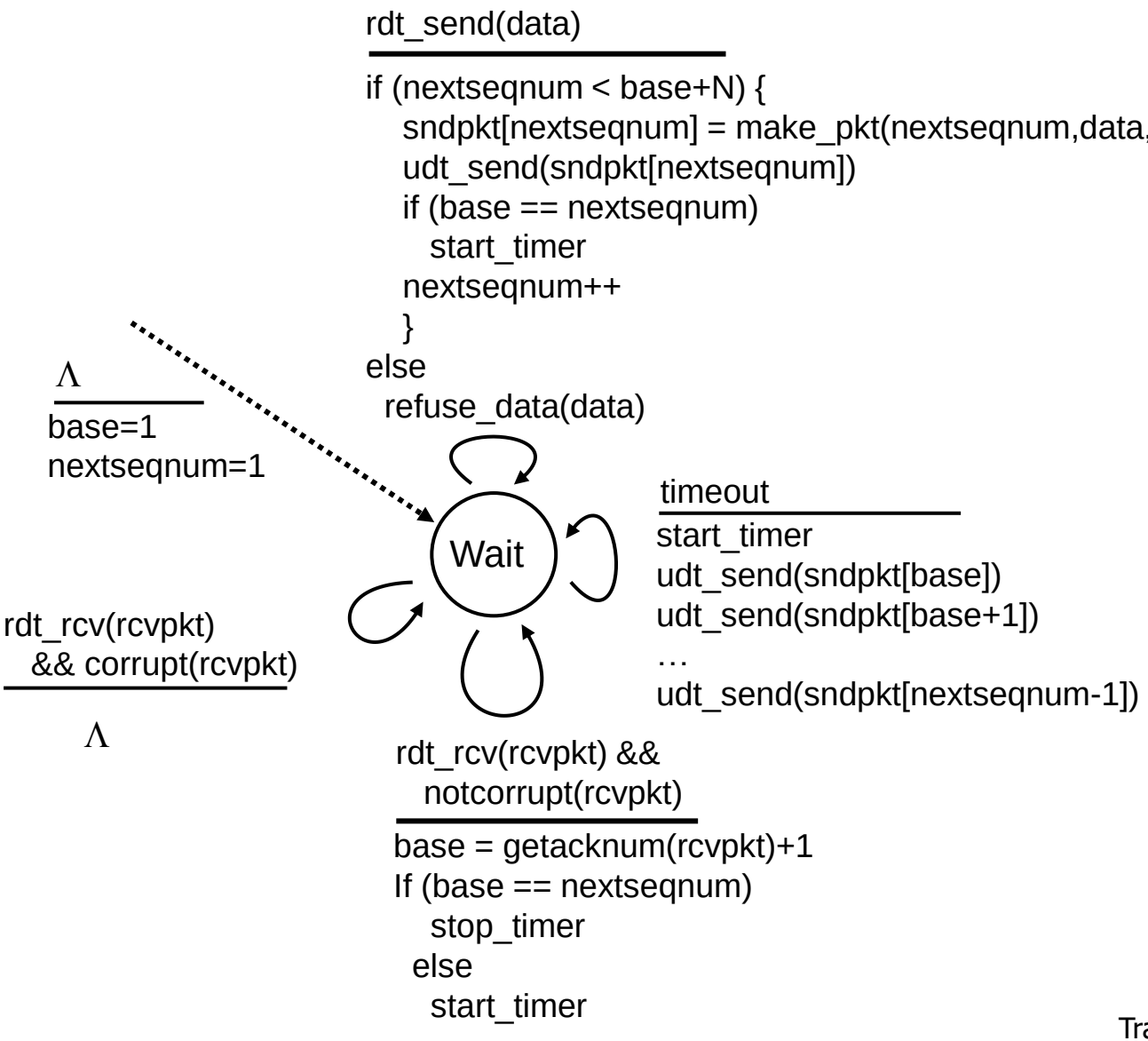
Go-Back-N: sender (Sliding Window)

- k-bit seq # in pkt header, range of sequence numbers is $[0, 2^k - 1]$
- sliding "window" of up to N, consecutive unack'ed pkts allowed
- N should be always greater than 1 in order to implement pipelining. For N = 1, it reduces to Stop and Wait protocol.(Min sequence numbers required in GBN=N+1)



- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"* (i.e. One ACK is used for many packets. The main advantage is that traffic is less. A disadvantage is less reliability i.e. if one ack is lost, that would mean that all the packets sent are lost.)
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

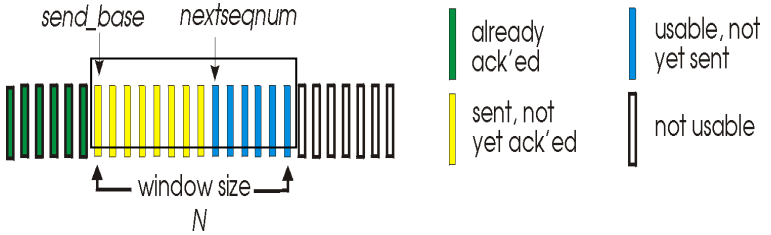
GBN: sender extended FSM (extended? Because variables similar to programming language variables are now added)



Go-Back-N: sender

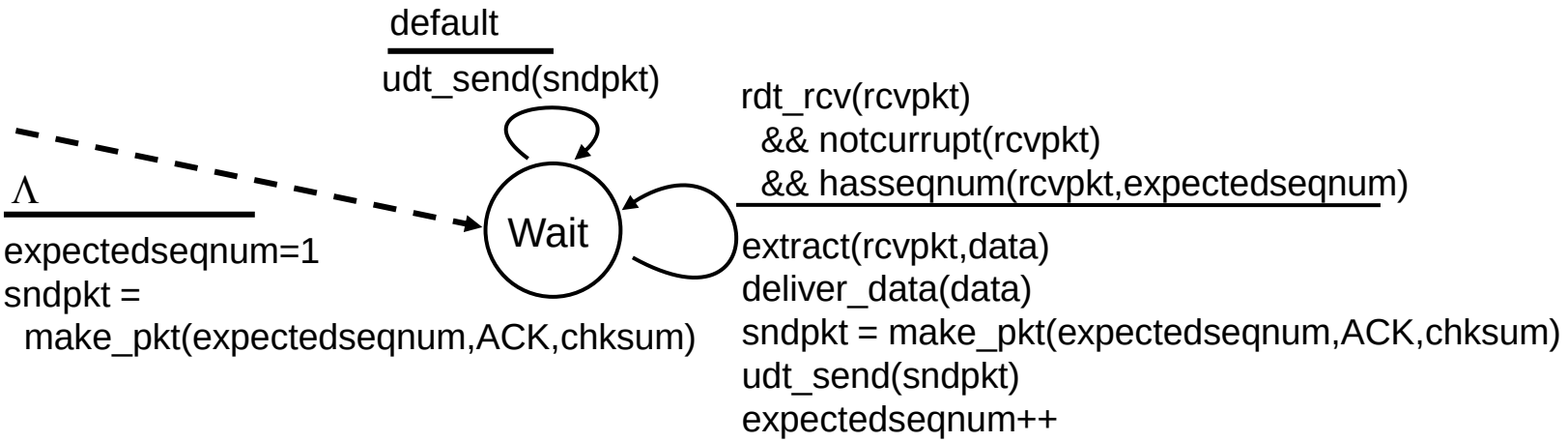
- Invocation from above
  - When rdt_sent() is called, checks if window is full
  - If not full, a packet is created and sent
- Receipt of an ACK
  - Cumulative acknowledgement: ack with seq number n means all previous packets has been received at receiver
- A timeout event
  - All previous packets that have been sent, their acknowledgements have not been received will be resent

4 Ranges of Sequence Numbers



1. [0, base -1] → are packets with sequence numbers sent & ACKed.
2. [base, nextseqnum - 1] → are sequence numbers sent but not yet ACKed (inflight.)
3. [nextseqnum, base + N - 1] → are sequence numbers that can be sent immediately if more data arrives from the App layer.
4. [>=, base + N] → are sequence numbers that can't be used until an unACKed packet is ACKed.

GBN: receiver extended FSM

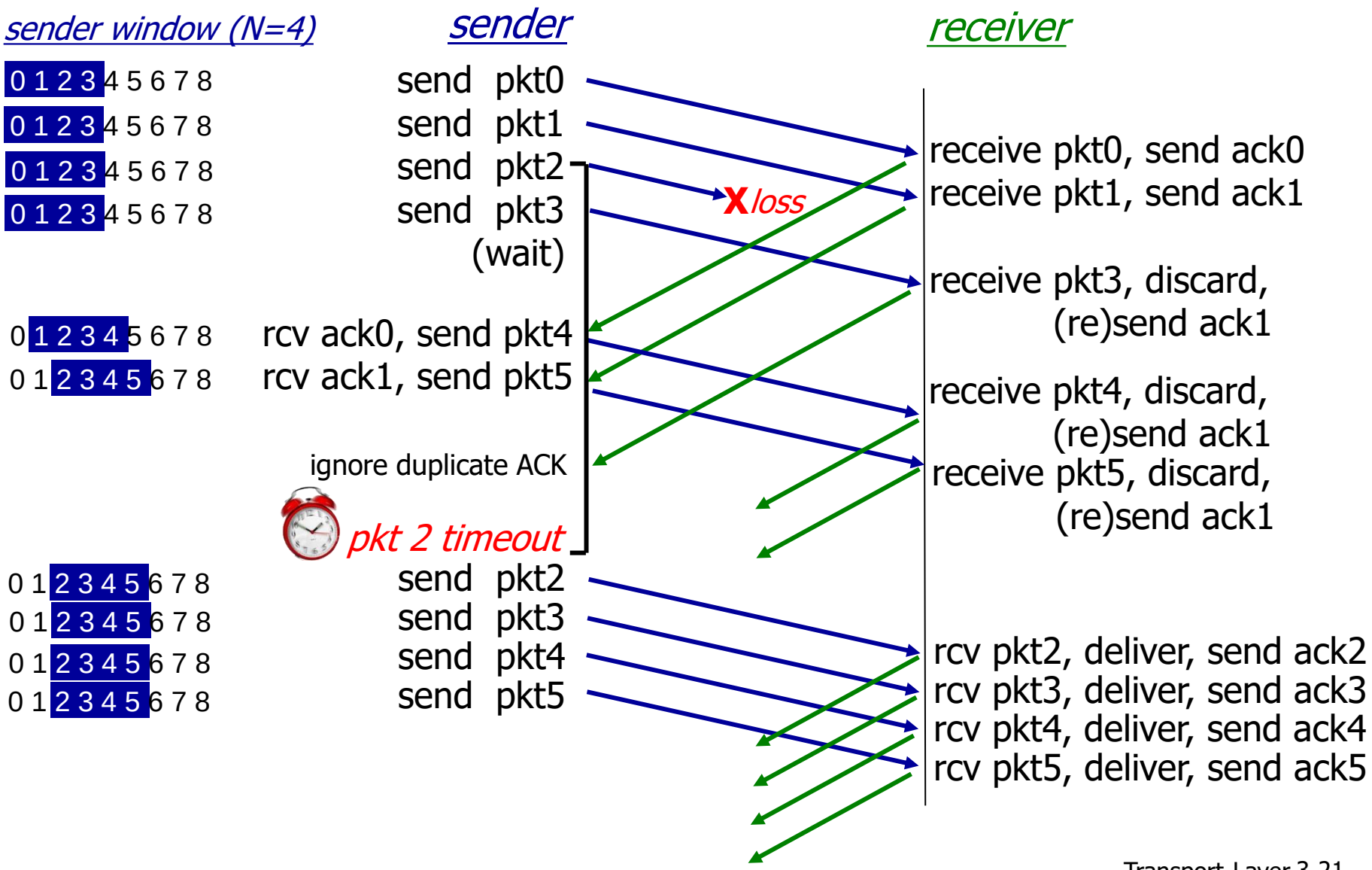


ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs (in case of corrupt packet received, it will resend the ACK of the previous correctly received packet)
- need to only remember **`expectedseqnum`**

- out-of-order pkt:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #
- Receiver window size is always 1

GBN in action



Midterm 1 Solution Discussion

- *Midterm 1 Solution was discussed*