Network Transport Layer: Sliding Window, TCP

Qiao Xiang, Congming Gao

https://sngroup.org.cn/courses/cnns-xmuf23/index.shtml

11/09/2023

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Outline

- Admin and recap
- Reliable data transfer

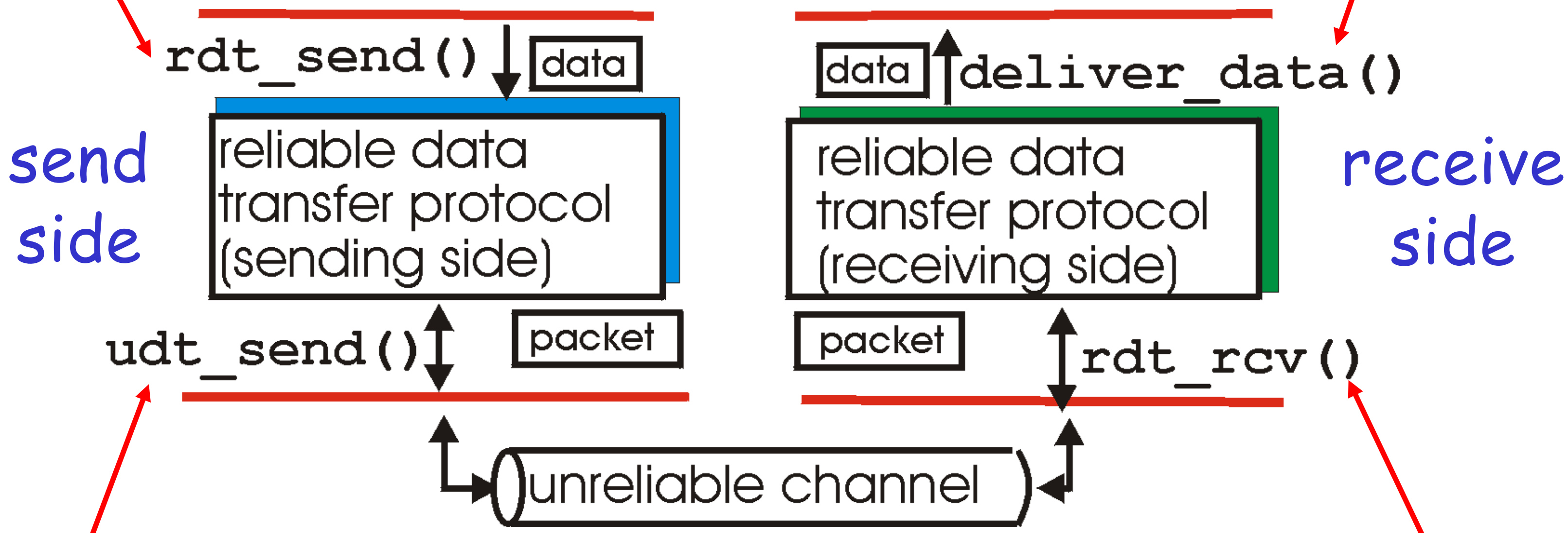
Admin

- Don't forget to bring your cheatsheet this afternoon

Recap: Reliable Data Transfer Context

rdt_send(): called from above, (e.g., by app.)

deliver_data(): called by **rdt** to deliver data to upper



udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

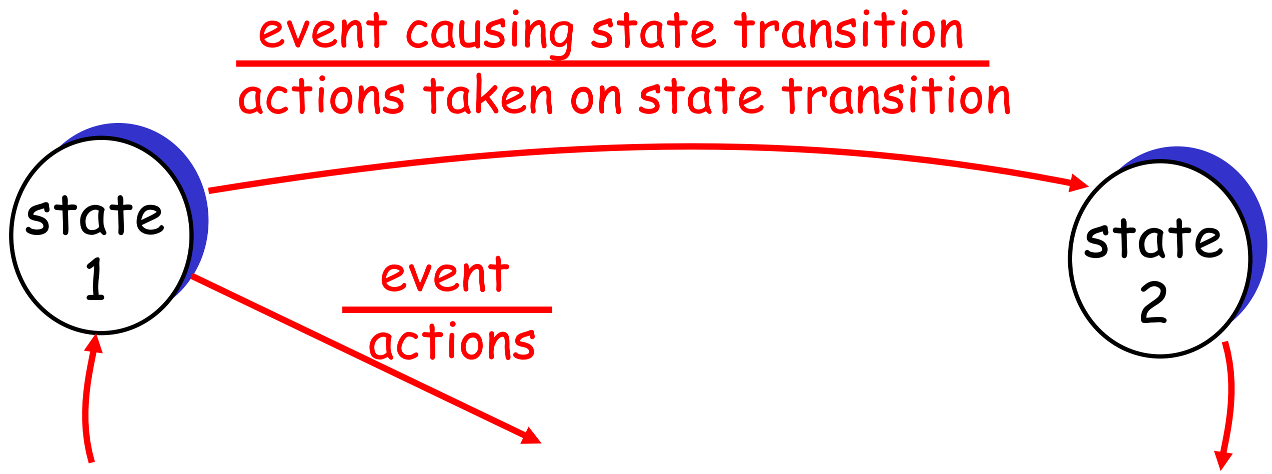
rdt_rcv(): called from below; when packet arrives on rcv-side of channel

Recap: Reliable Data Transfer Setting

We'll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions !
- use finite state machines (FSM) to specify sender, receiver

state: when in this "state" next state uniquely determined by next event



rdt3.0: Channels with Errors *and* Loss

New assumption: underlying channel can also lose packets (data or ACKs)

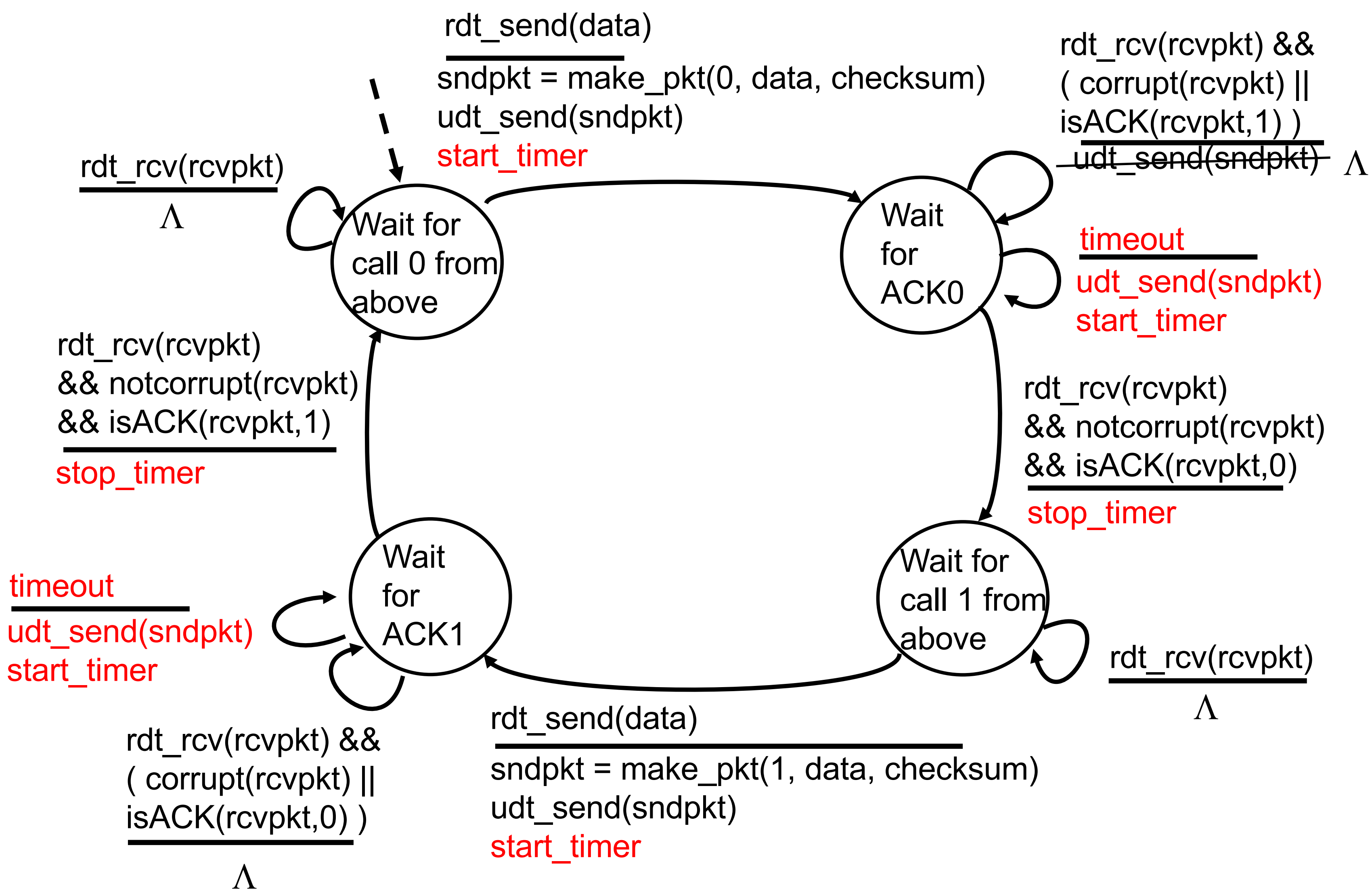
- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

Q: Does rdt2.2 work under losses?

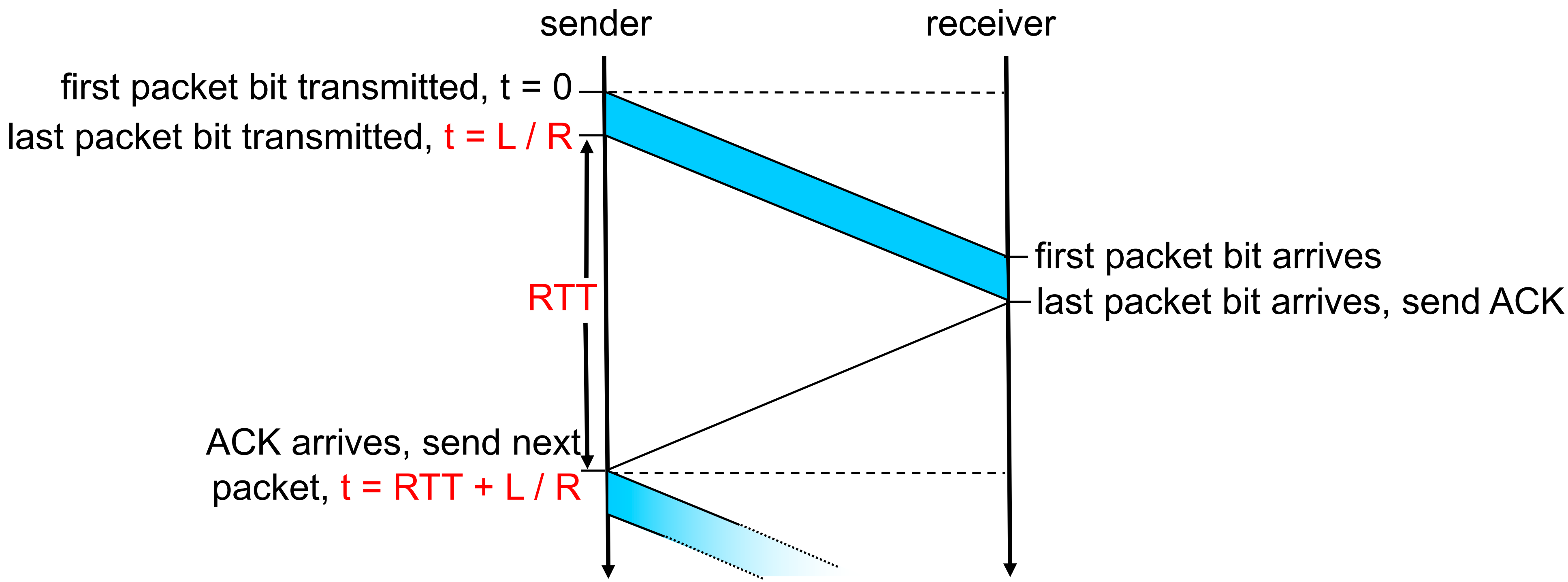
Approach: sender waits "reasonable" amount of time for ACK

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

rdt3.0 Sender



rdt3.0: Stop-and-Wait Performance



What is U_{sender}: utilization – fraction of time link busy sending?

Assume: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet

Performance of rdt3.0

- rdt3.0 works, but performance stinks
- Example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8\text{kb/pkt}}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$

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

- 1KB pkt every 30 msec -> 33kB/sec throughput over 1 Gbps link
- network protocol limits use of physical resources !

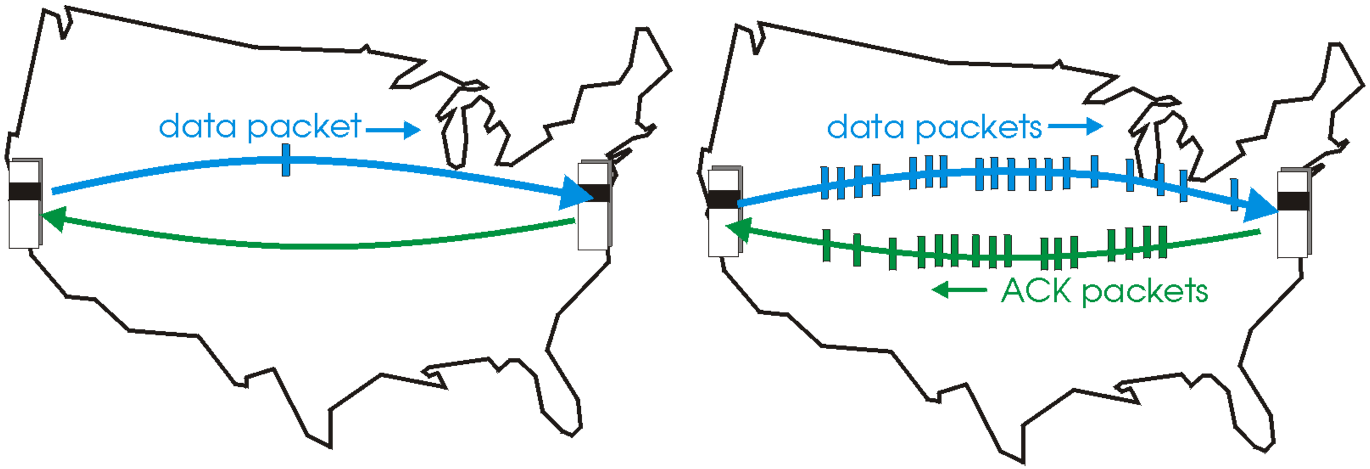
A Summary of Questions

- How to improve the performance of rdt3.0?
- What if there are reordering and duplication?
- How to determine the "right" timeout value?

Sliding Window Protocols: Pipelining

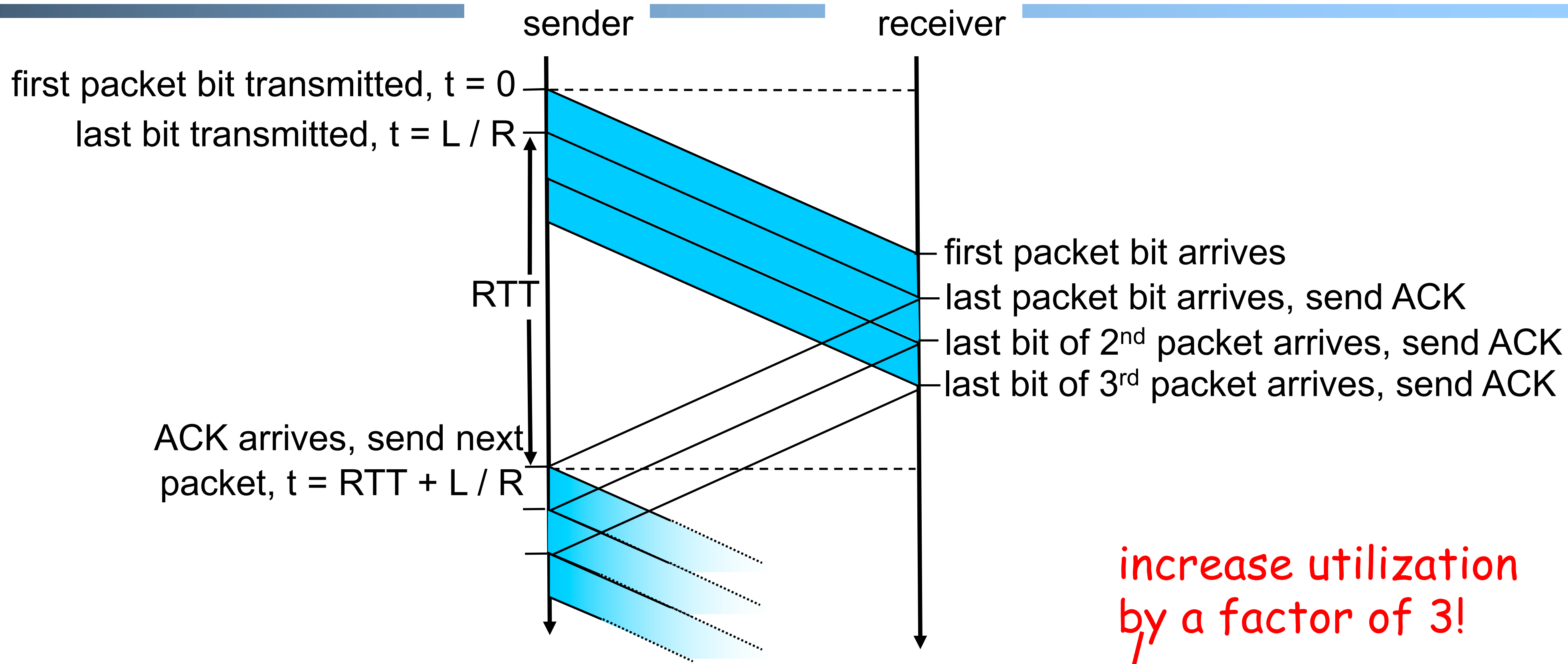
Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

- 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

Pipelining: Increased Utilization



increase utilization by a factor of 3!

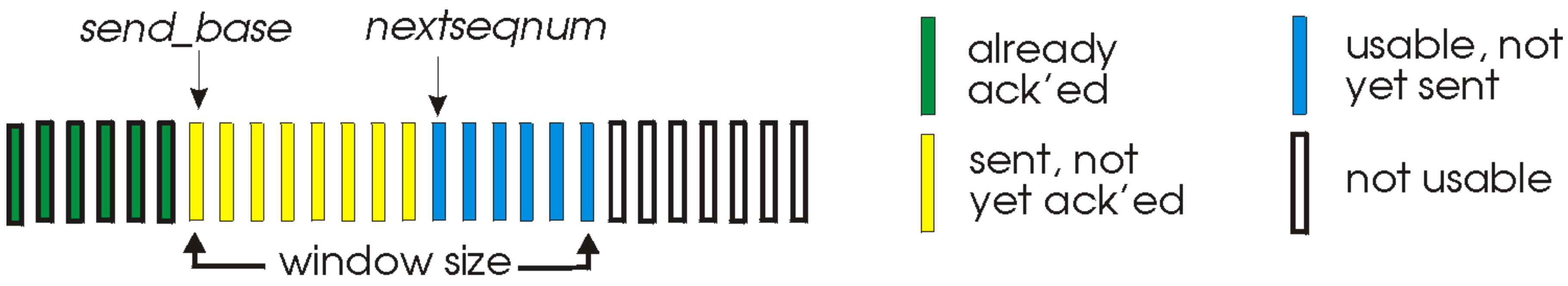
$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Question: a rule-of-thumb window size?

Realizing Sliding Window: Go-Back-n

Sender:

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



W

- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - note: ACK(n) could mean two things: I have received upto and include n, or I am waiting for n
- timer for the packet at base
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

GBN: Sender FSM

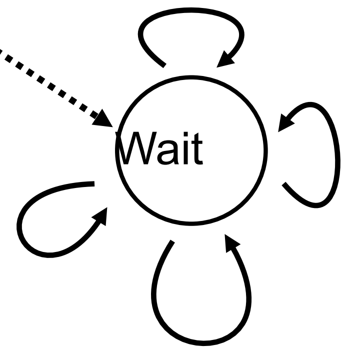
rdt_send(data)

```
if (nextseqnum < base+W) {
    sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
    udt_send(sndpkt[nextseqnum])
    if (base == nextseqnum) start_timer
    nextseqnum++
 } else
    block sender
```

Λ

---

base=1
nextseqnum=1



timeout

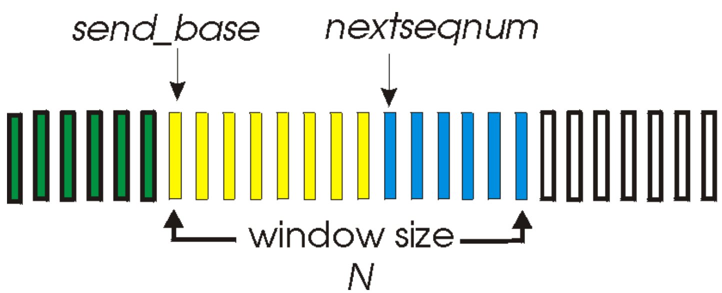
```
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
…
udt_send(sndpkt[nextseqnum-1])
```

rdt_rcv(rcvpkt)
&& corrupt(rcvpkt)

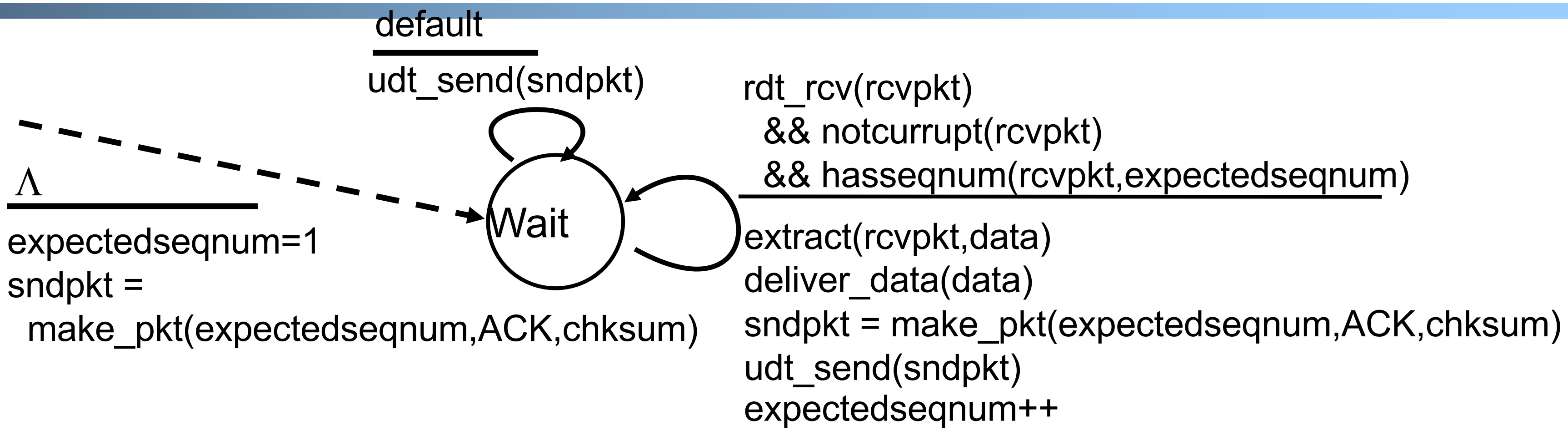
---

rdt_rcv(rcvpkt) &&
notcorrupt(rcvpkt)

```
if (new packets ACKed) {
   advance base;
   if (more packets waiting)
       send more packets
}
if (base == nextseqnum)
  stop_timer
else
  start_timer for the packet at new base
```



GBN: Receiver FSM

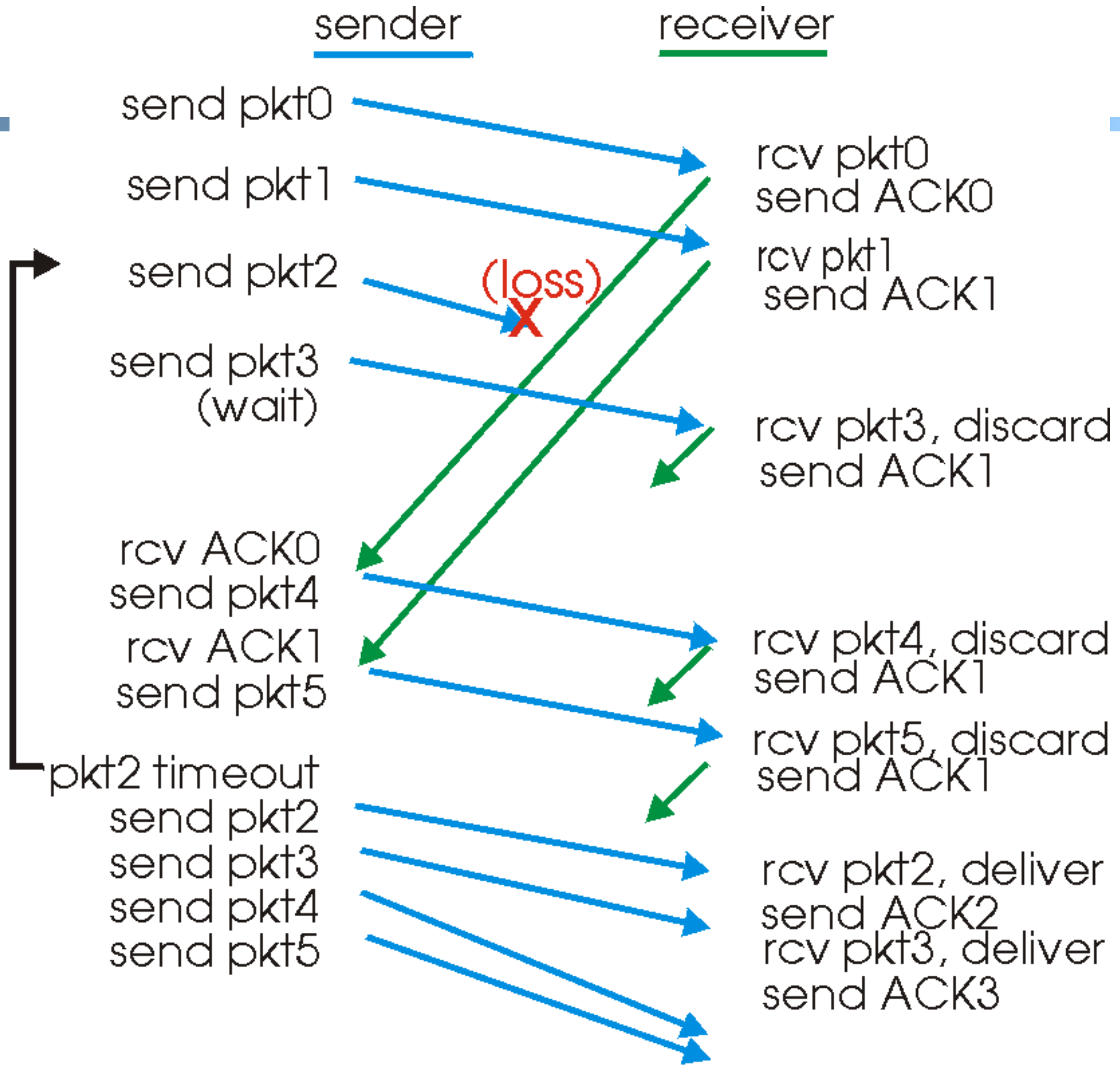


Only state: **`expectedseqnum`**

- out-of-order pkt:
  - discard (don't buffer) -> no receiver buffering!
  - re-ACK pkt with highest in-order seq #
  - may generate duplicate ACKs

GBN in Action

window
size = 4



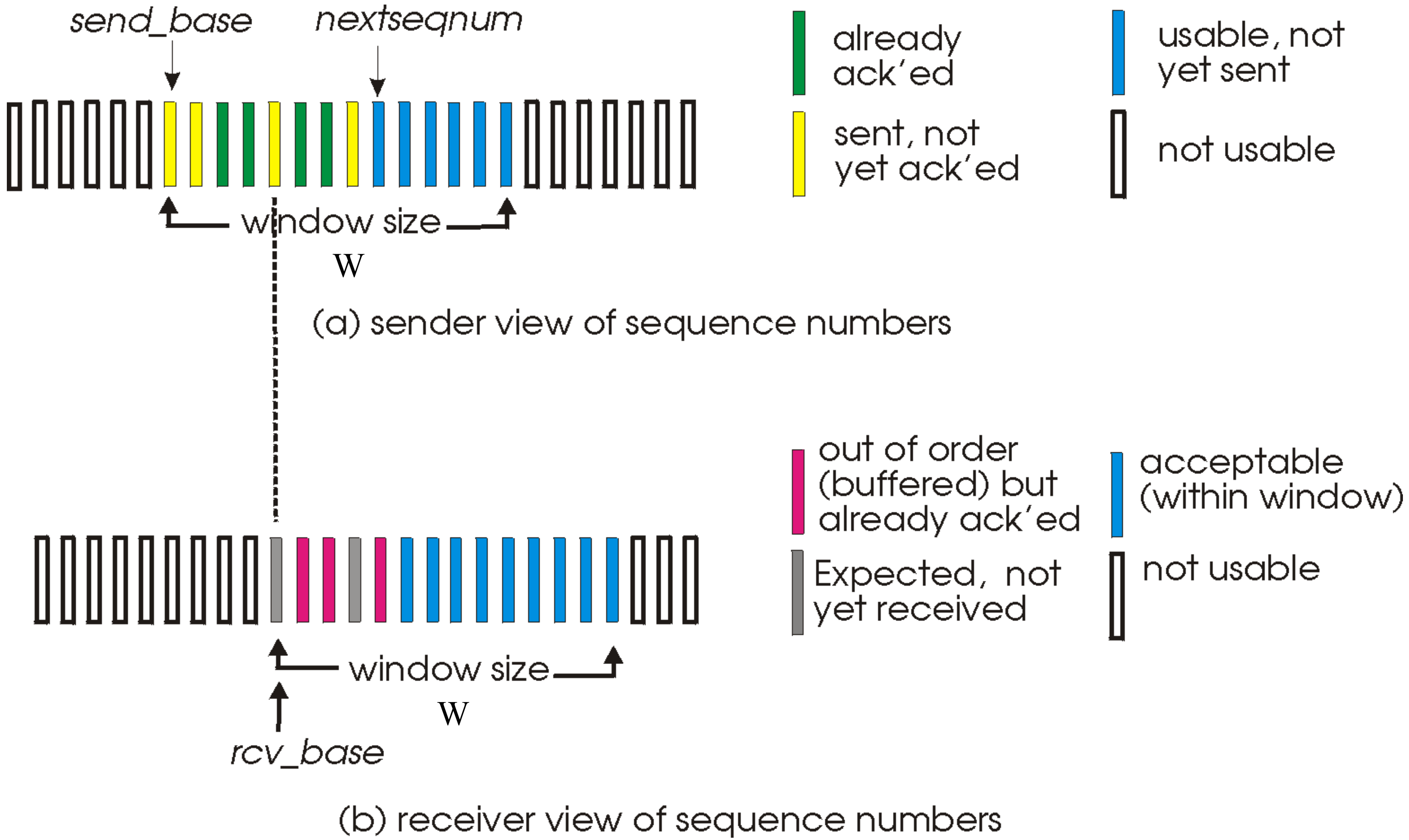
Analysis: Efficiency of Go-Back-n

- Assume window size W
- Assume each packet is lost with probability p
- On average, how many packets do we send for each data packet received?

Selective Repeat

- Sender window
  - Window size W: W consecutive unACKed seq #'s
- Receiver *individually* acknowledges correctly received pkts
  - buffers out-of-order pkts, for eventual in-order delivery to upper layer
  - ACK(n) means received packet with seq# n only
  - buffer size at receiver: window size
- Sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt

Selective Repeat: Sender, Receiver Windows



Selective Repeat

sender

data from above :

- unACKed packets is less than window size W, send; otherwise block app.

timeout(n):

- resend pkt n, restart timer

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

- mark pkt n as received
- update sendbase to the first packet unACKed

receiver

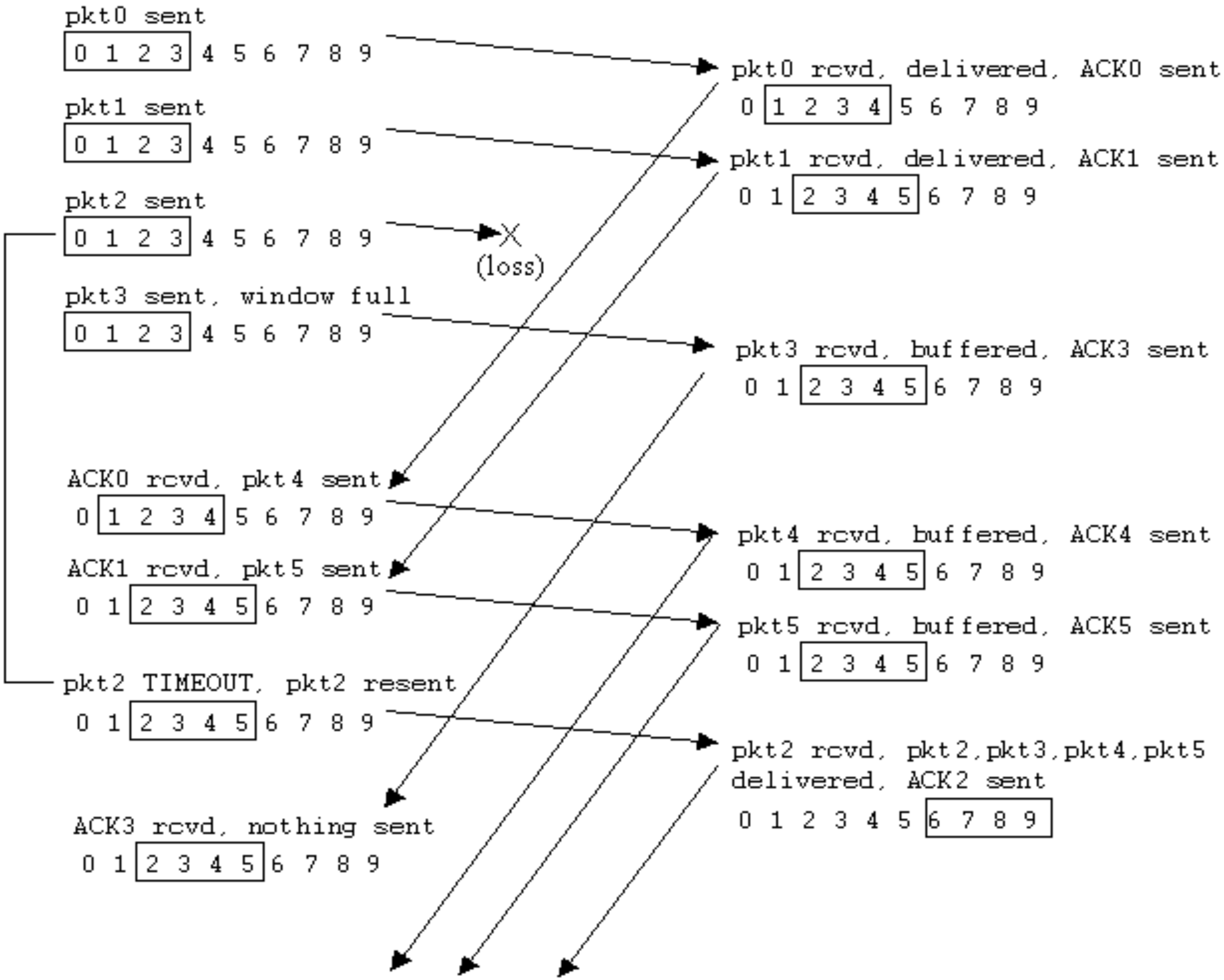
pkt n in [rcvbase, rcvbase+W-1]

- send ACK(n)
- if (out-of-order)
  mark and buffer pkt n
  else /*in-order*/
  deliver any in-order packets

otherwise:

- ignore

Selective Repeat in Action



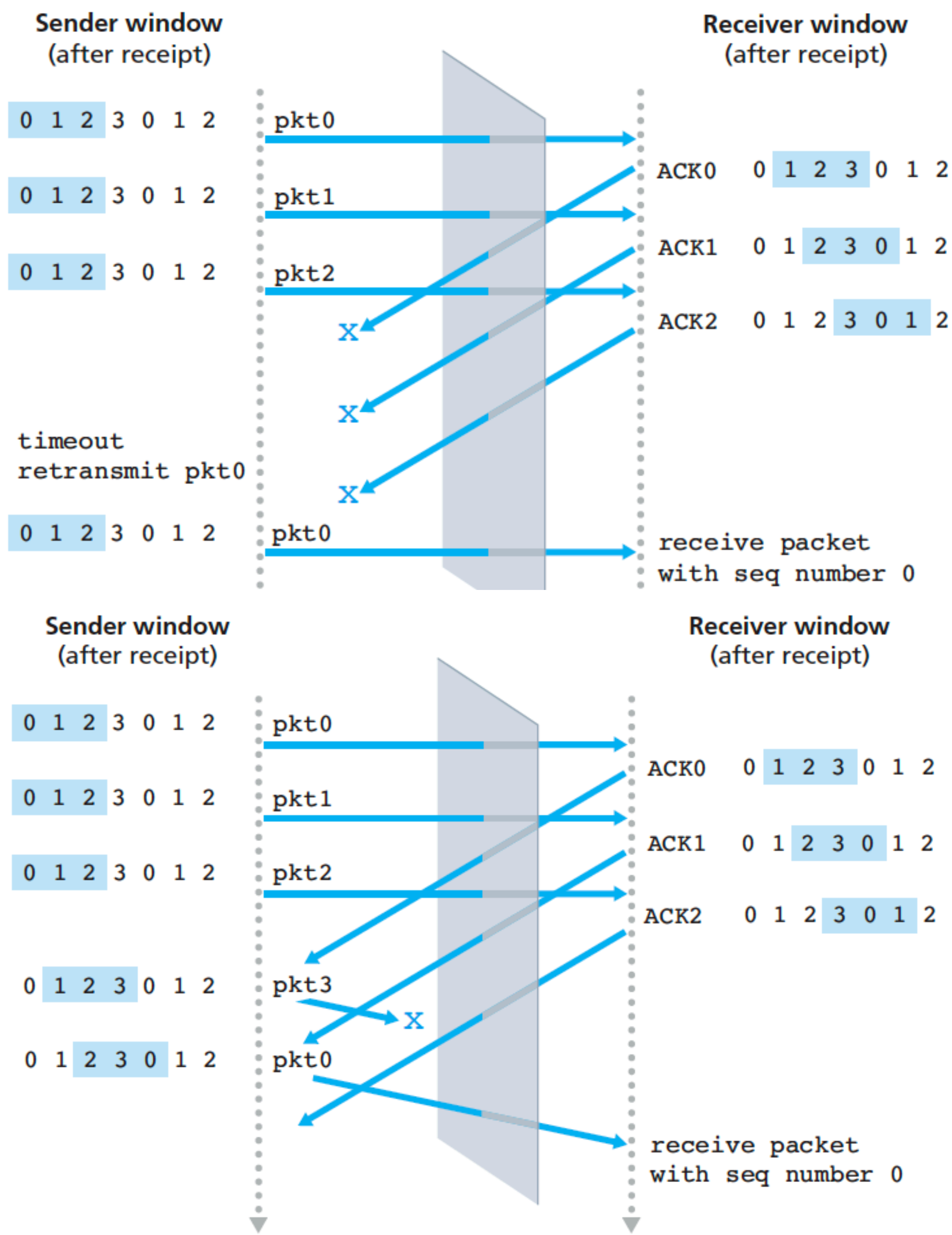
Discussion: Efficiency of Selective Repeat

- Assume window size W
- Assume each packet is lost with probability p
- On average, how many packets do we send for each data packet received?

Selective Repeat: Seq# Ambiguity

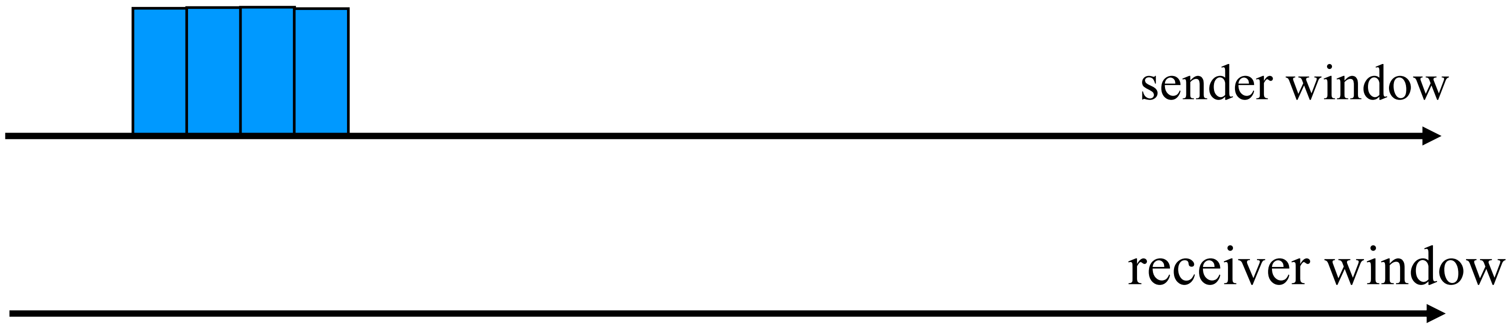
Example:
- seq #'s: 0, 1, 2, 3
- window size=3

- Error: incorrectly passes duplicate data as new.

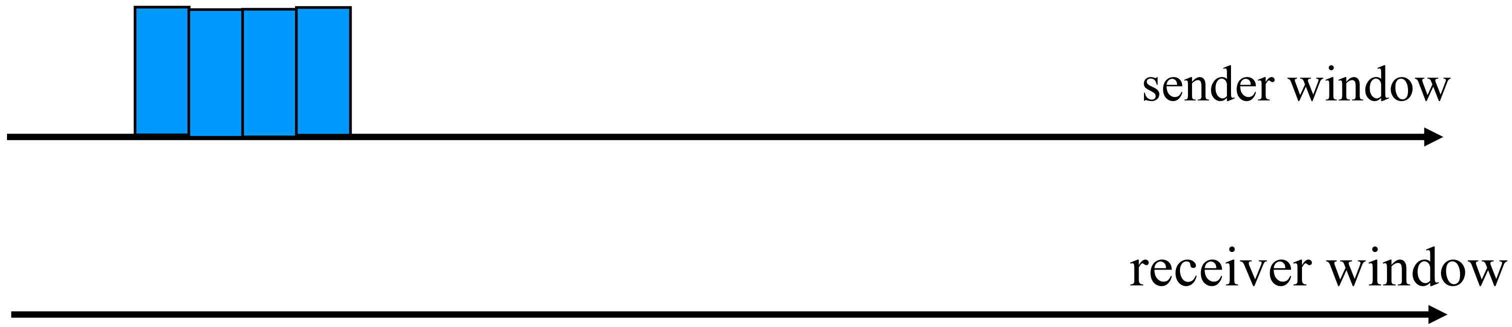


State Invariant: Window Location

- Go-back-n (GBN)



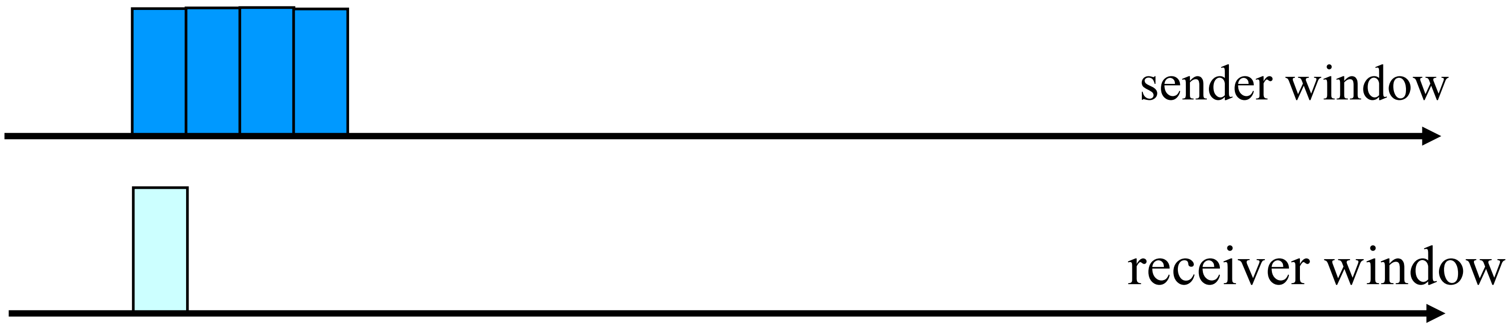
- Selective repeat (SR)



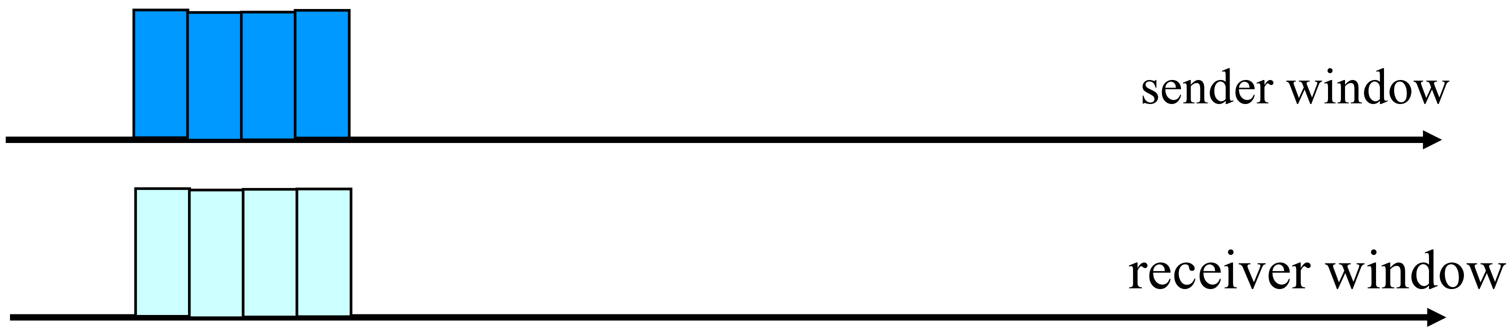
Window Location

Q: what relationship between seq # size and window size?

- Go-back-n (GBN)



- Selective repeat (SR)



Selective Repeat

sender

data from above :

- unACKed packets is less than window size W, send; otherwise block app.

timeout(n):

- resend pkt n, restart timer

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

- mark pkt n as received
- update sendbase to the first packet unACKed

receiver

pkt n in [rcvbase, rcvbase+W-1]

- send ACK(n)
- if (out-of-order)
  mark and buffer pkt n
  else /*in-order*/
  deliver any in-order packets

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

- send ACK(n)

otherwise:

- ignore

Sliding Window Protocols: Go-back-n and Selective Repeat

	Go-back-n	Selective Repeat
data bandwidth: sender to receiver (avg. number of times a pkt is transmitted)	Less efficient $\frac{1-p+pw}{1-p}$	More efficient $\frac{1}{1-p}$
ACK bandwidth (receiver to sender)	More efficient	Less efficient
Relationship between M (the number of seq#) and W (window size)	M > W	M ≥ 2W
Buffer size at receiver	1	W
Complexity	Simpler	More complex

p: the loss rate of a packet; M: number of seq# (e.g., 3 bit M = 8); W: window size

Outline

- Admin and Recap
- Reliable data transfer
  - perfect channel
  - channel with bit errors
  - channel with bit errors and losses
  - sliding window: reliability with throughput
- *TCP reliability*

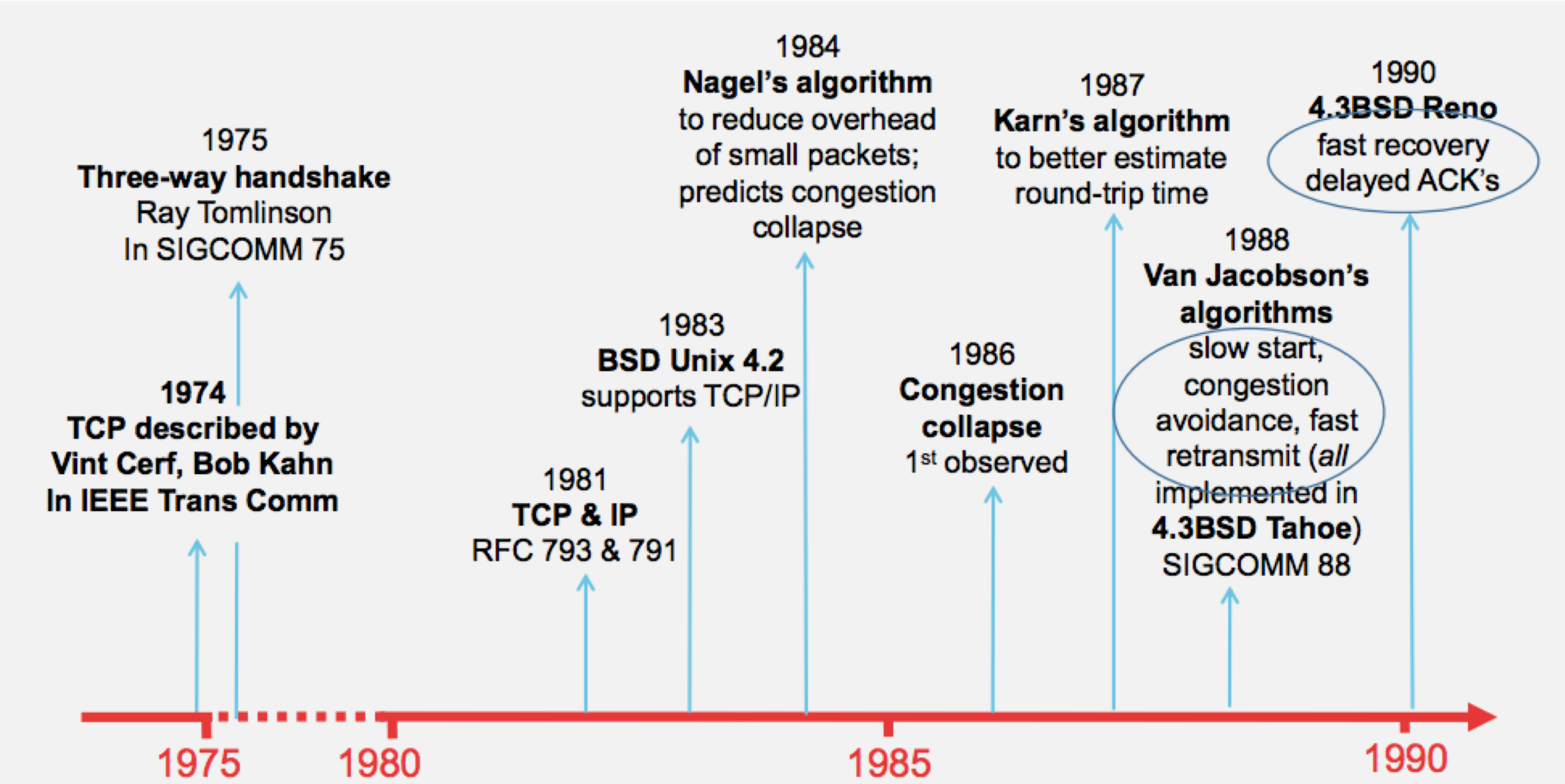
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- Point-to-point reliability: one sender, one receiver

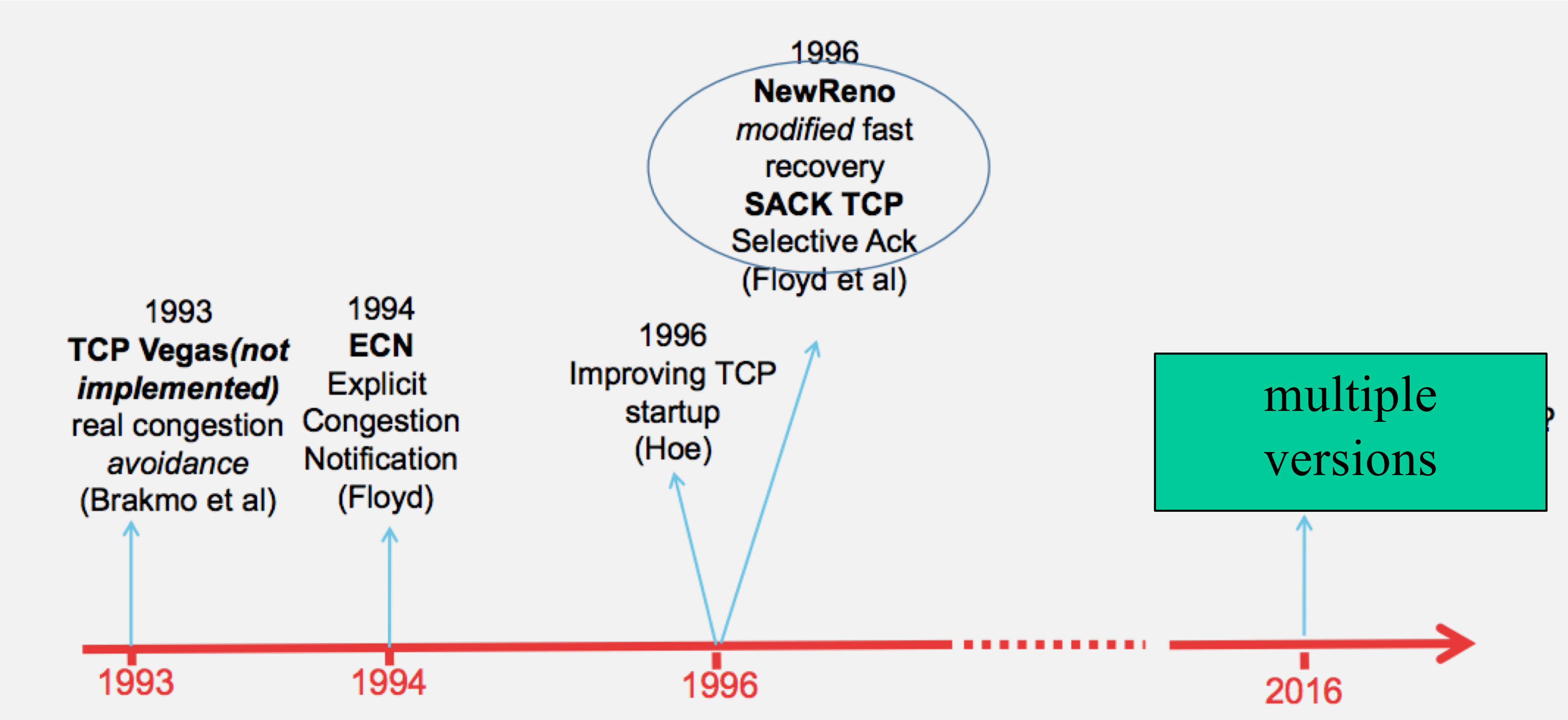
- Flow controlled and congestion controlled

Evolution of TCP



Source: http://webcourse.cs.technion.ac.il/236341/Winter2015-2016/ho/WCFiles/Tutorial10.pdf

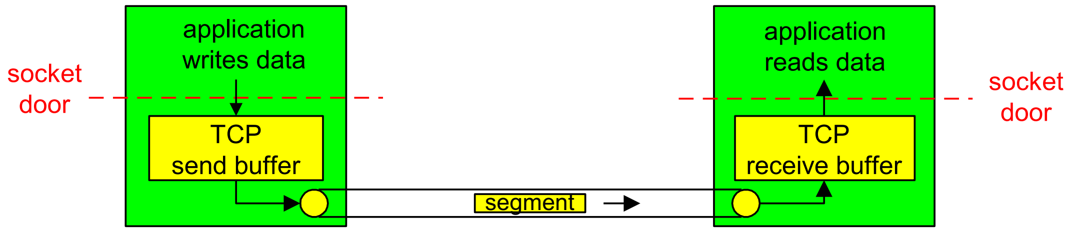
Evolution of TCP



Source: http://webcourse.cs.technion.ac.il/236341/Winter2015-2016/ho/WCFiles/Tutorial10.pdf

TCP Reliable Data Transfer

- **Connection-oriented:**
  - connection management
    - setup (exchange of control msgs) init's sender, receiver state before data exchange
    - close
- **Full duplex data:**
  - bi-directional data flow in same connection
- **A sliding window protocol**
  - a combination of go-back-n and selective repeat:
    - send & receive buffers
    - cumulative acks
    - TCP uses a single retransmission timer
    - do not retransmit all packets upon timeout



TCP Segment Structure