Performance of rdt3.0

- rdt3.0 works, but performance could be desastrous

Example:

- 1 Gbps link, i.e. transmission rate of 10^9 bits per second
- 15 ms propagation delay,
- 1KB frame length, i.e. 8 000 bits per frame

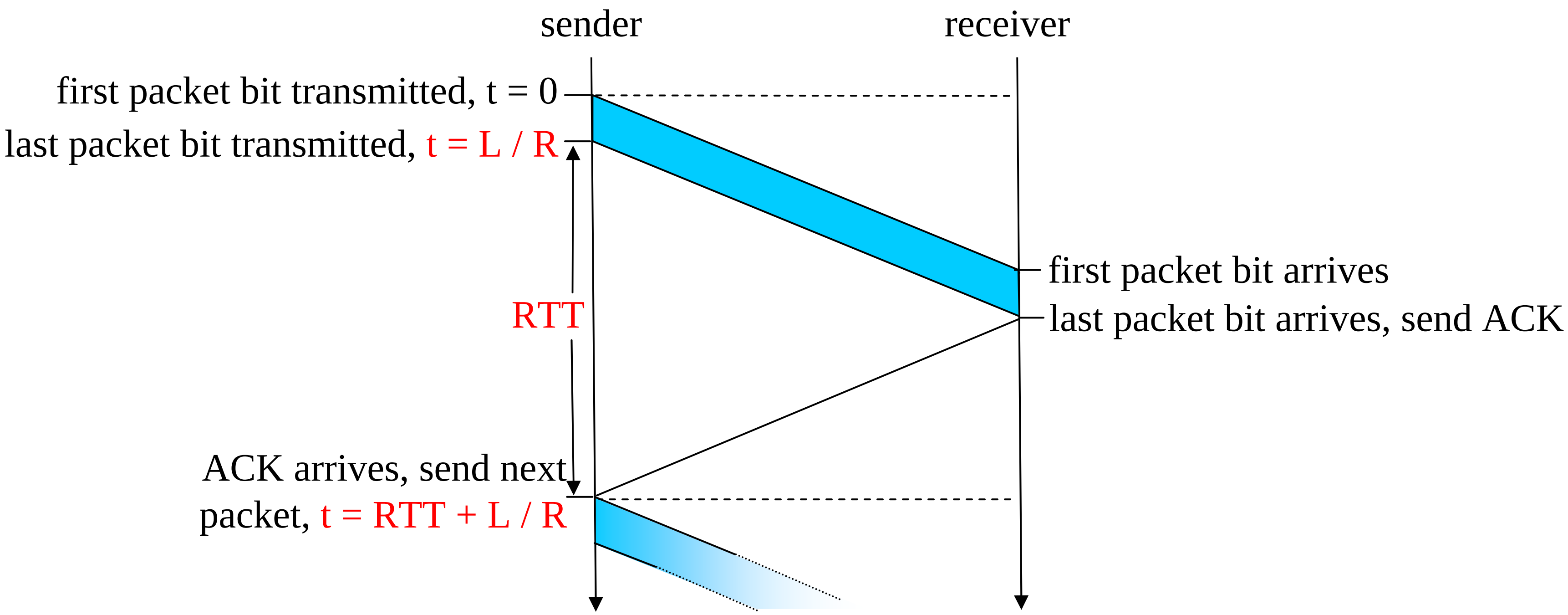
$$T_{\text{Transmission delay}} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8\text{kb/pkt}}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$

- $U_{\text{sender/channel}}$: utilization – fraction of time sender busy sending
- RTT: Round-Trip Time

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

- 1KB pkt every 30 msec -> 267kb/sec throughput over 1 Gbps link
- network protocol limits use of physical resources!

rdt3.0: stop-and-wait operation



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

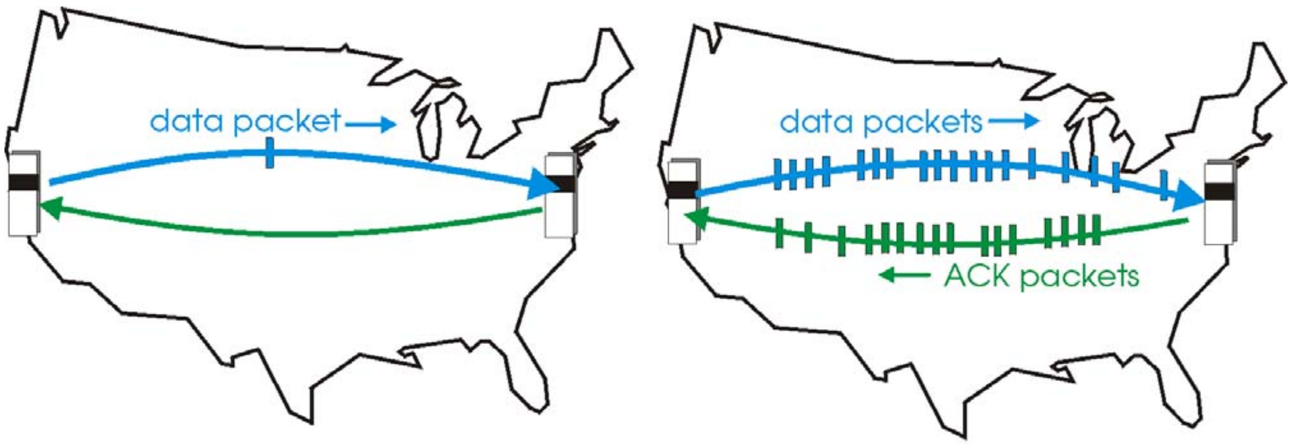
Implicit assumption so far:

Propagation delay of the medium is negligible or its bandwidth (transmission rate) is very low

If this assumption is false ---> exploitation of the bandwidth may be disastrous ---> requiring a sender to wait for an ack for each single frame before sending the next frame must be relaxed

Pipelined protocols

Pipelining: sender allows multiple, “in-transit”, yet-to-be-acknowledged frames, the sender is allowed to transmit up to *N* frames before blocking, instead of just 1.



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

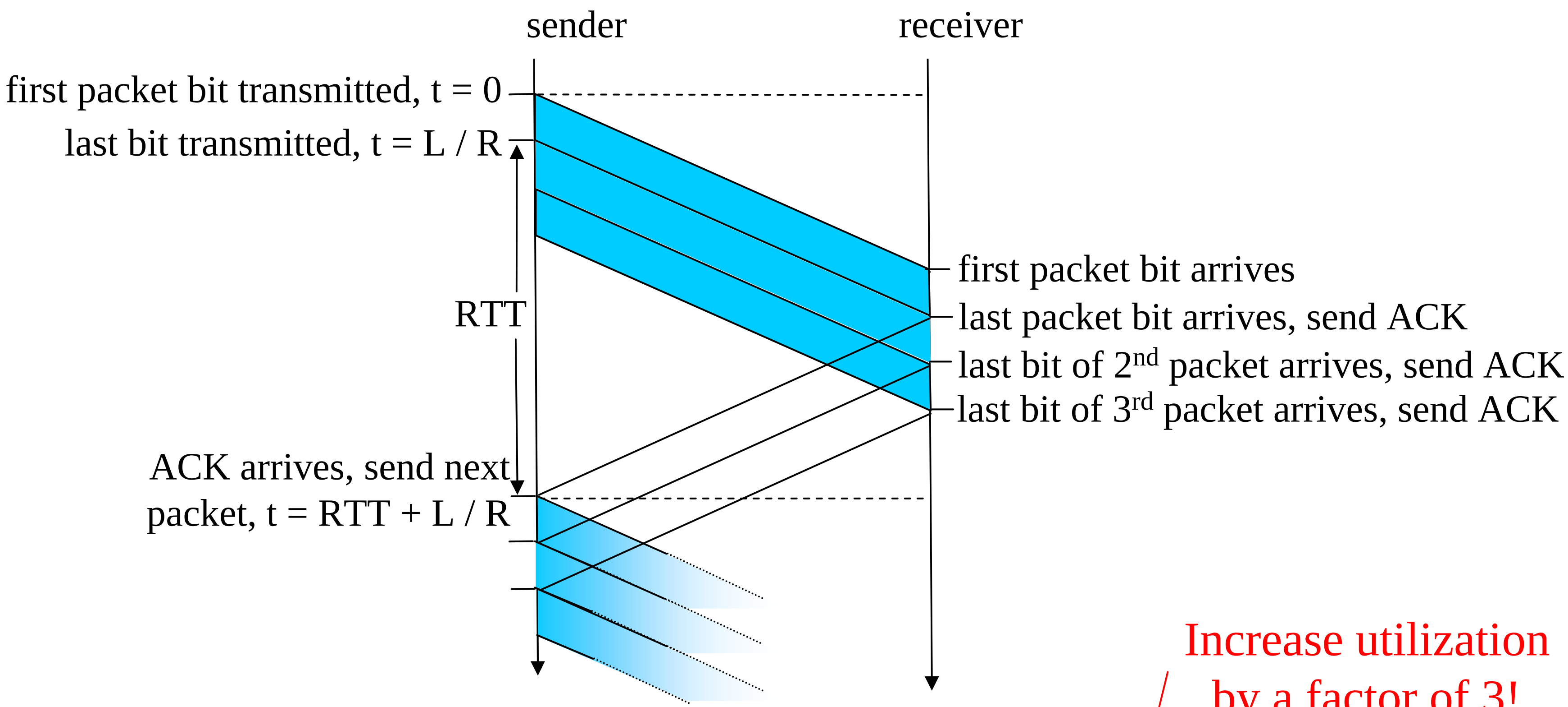
Consequences:

- range of sequence numbers must be increased
- buffering at sender

What exactly happens if a frame is lost or damaged in the middle of a long stream of transmitted frames?

Two basic approaches : *go-Back-N, selective repeat*

Pipelining: increased utilization



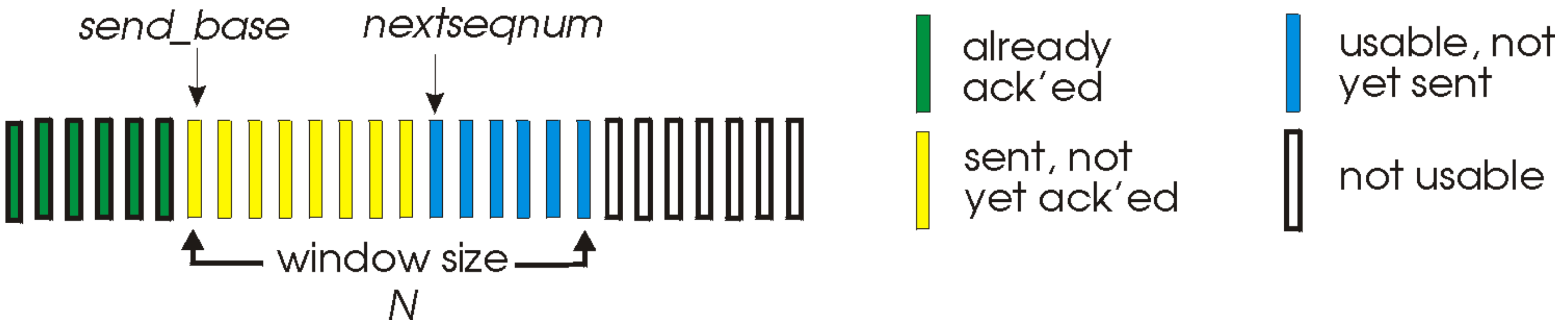
Increase utilization by a factor of 3!

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

Go-Back-N

Sender:

- k-bit seq # in frame header
- “window” of up to $N = 2^k$ consecutive unack’ed frames allowed



- timer for each in-transit frame
- *timeout(n):* retransmit frame and all higher seq # frames in window already sent

Receiver:

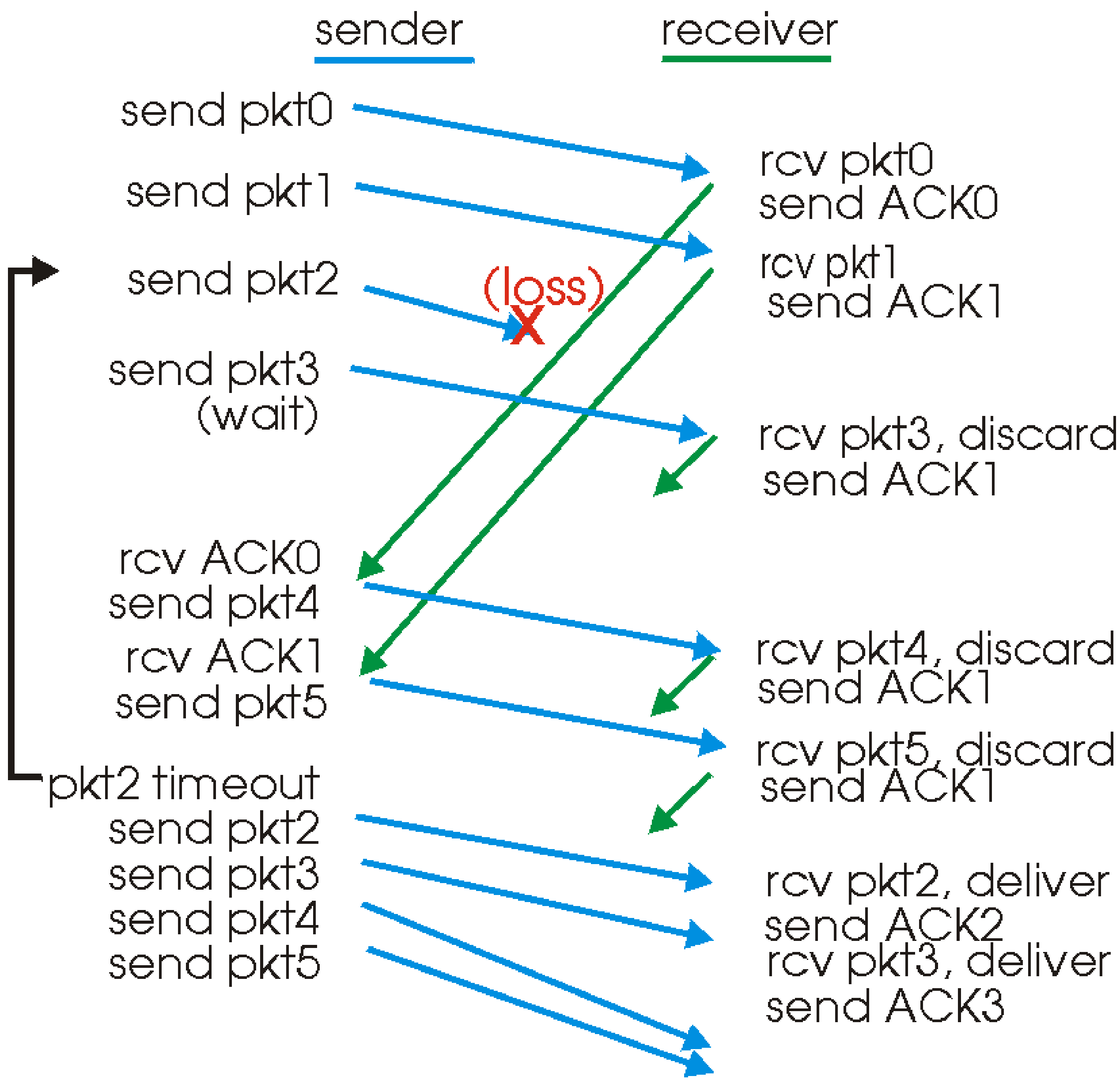
- ACK(n): ACKs all frames up to, including seq # n - “cumulative ACK”
- All frames arriving after an erroneous one are simply discarded, i.e the receiving entity refuses to accept any frame except the next one to be delivered to the network layer

  ---> eventually, the sender will time out and retransmit all unacknowledged frames in order starting with the erroneous one

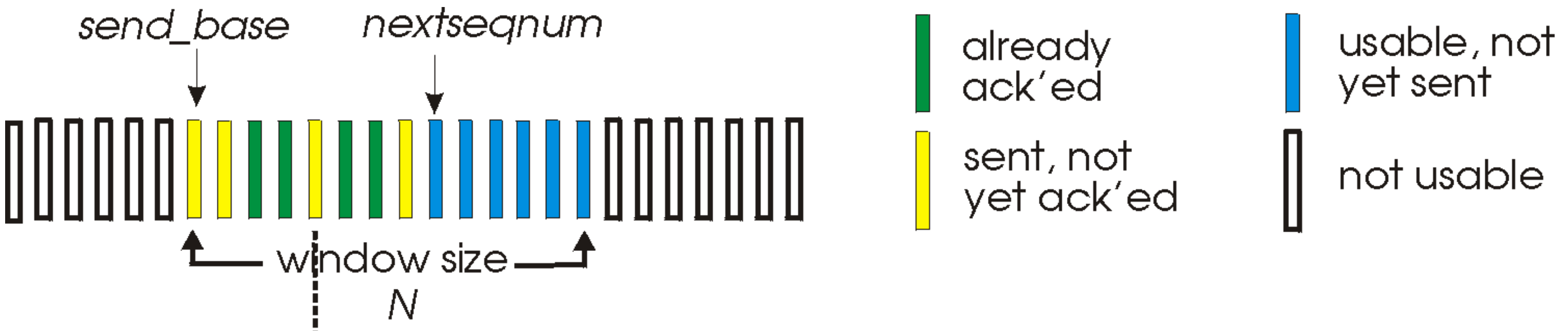
This strategy corresponds to a send window of size *N* and a receive window of size *1*.

Main drawback: It can waste a lot of bandwidth if the error rate is high

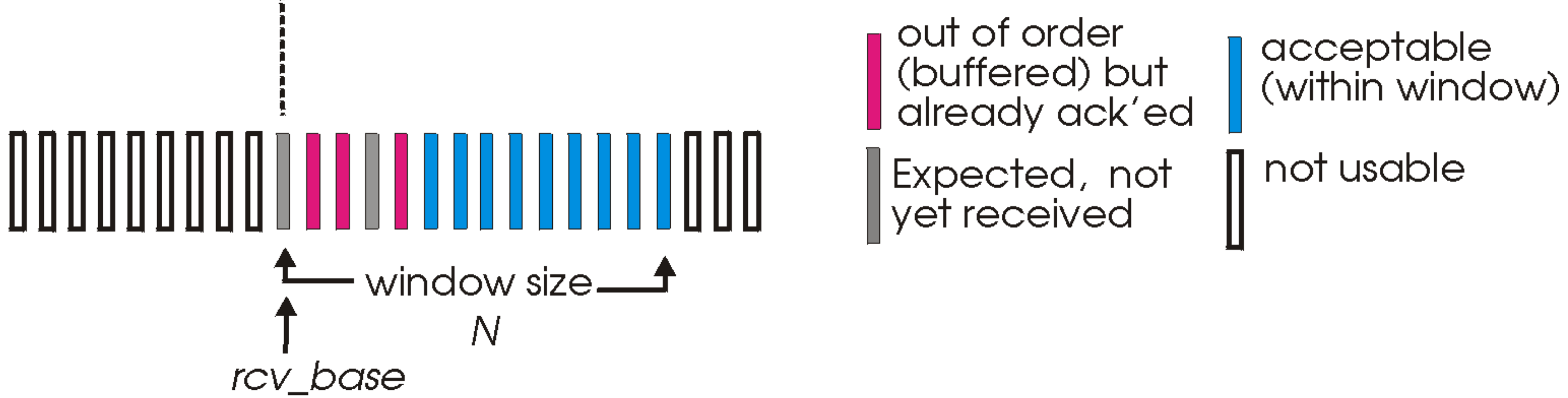
Go-Back-N (example)



Selective repeat: send and receive windows

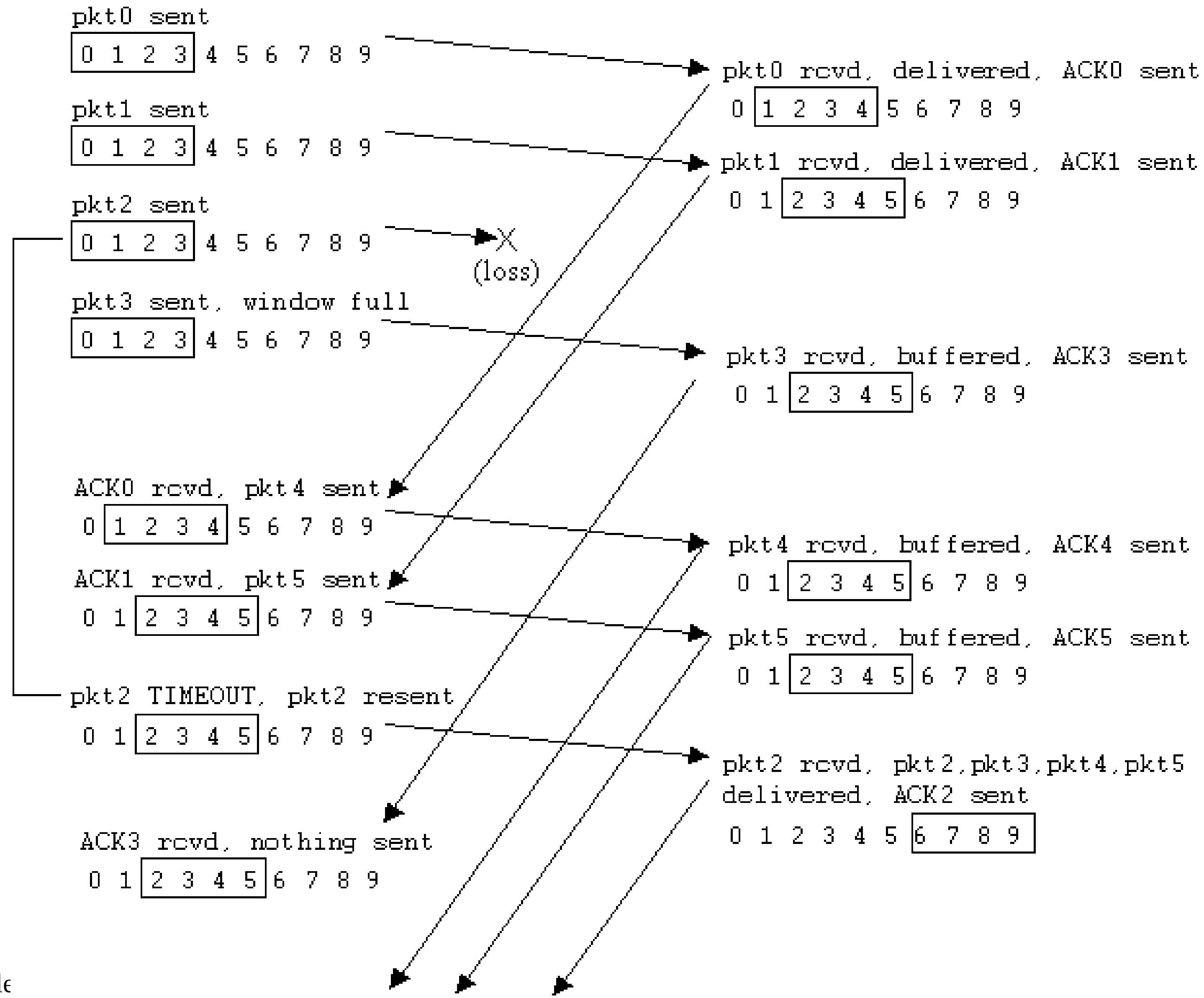


(a) sender view of sequence numbers



(b) receiver view of sequence numbers

Selective repeat in action (Example)



Selective Repeat

- receiver *individually* acknowledges all correctly received frames, i.e. all correct frames arriving after an erroneous one are accepted by the receiver as long as they fit into the receiver buffer

  ---> must buffer frames, as needed, for eventual in-order delivery to upper layer
- sender resends frames for which ACK not received
  - sender must set a timer for each unACKed frame
- send window
  - *N* consecutive seq #'s
  - again limits seq #s of sent unACKed frames
- receive window also of size *N*
- Main drawback: It can require large amounts of data link layer buffer space

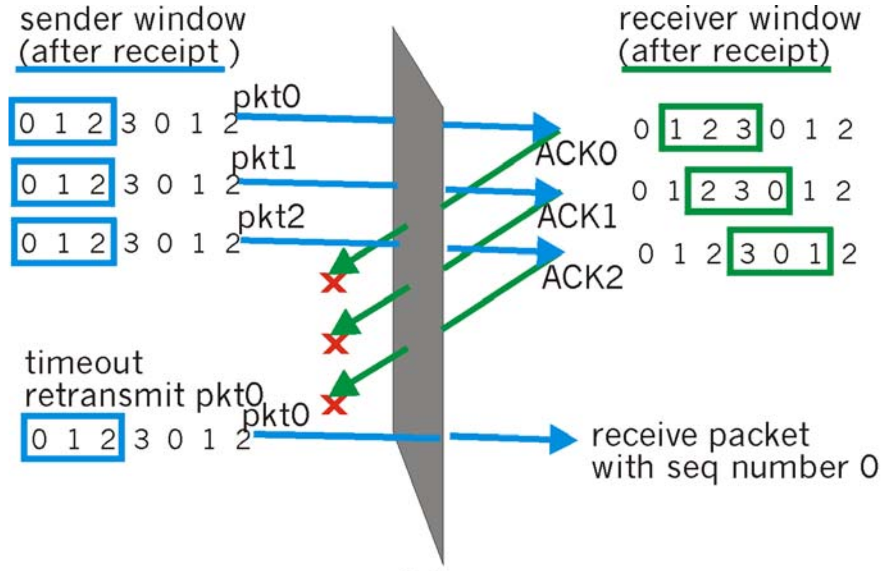
Selective repeat: dilemma

Example:

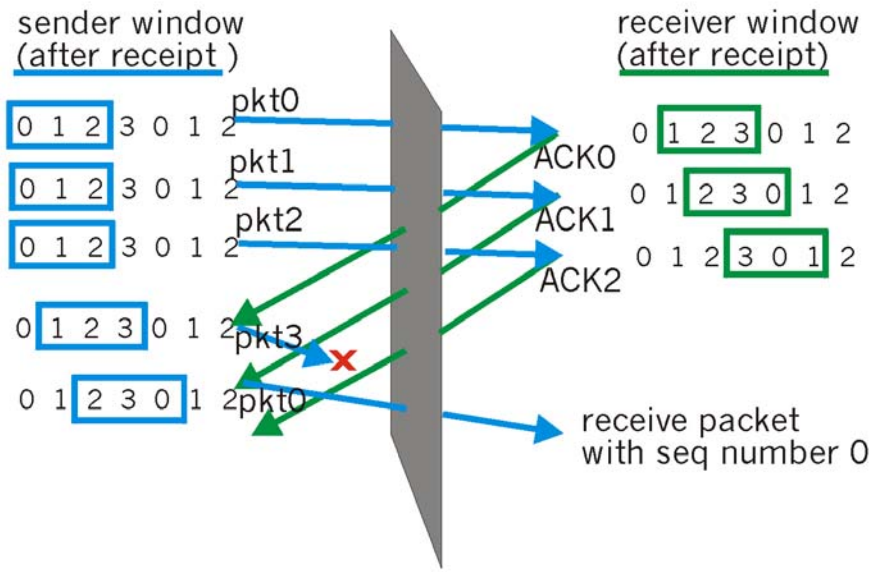
- seq #'s: 0, 1, 2, 3
- window size=3

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

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



(a)



(b)

Data Link Layer(18)

Pipelining implies multiple outstanding frames.

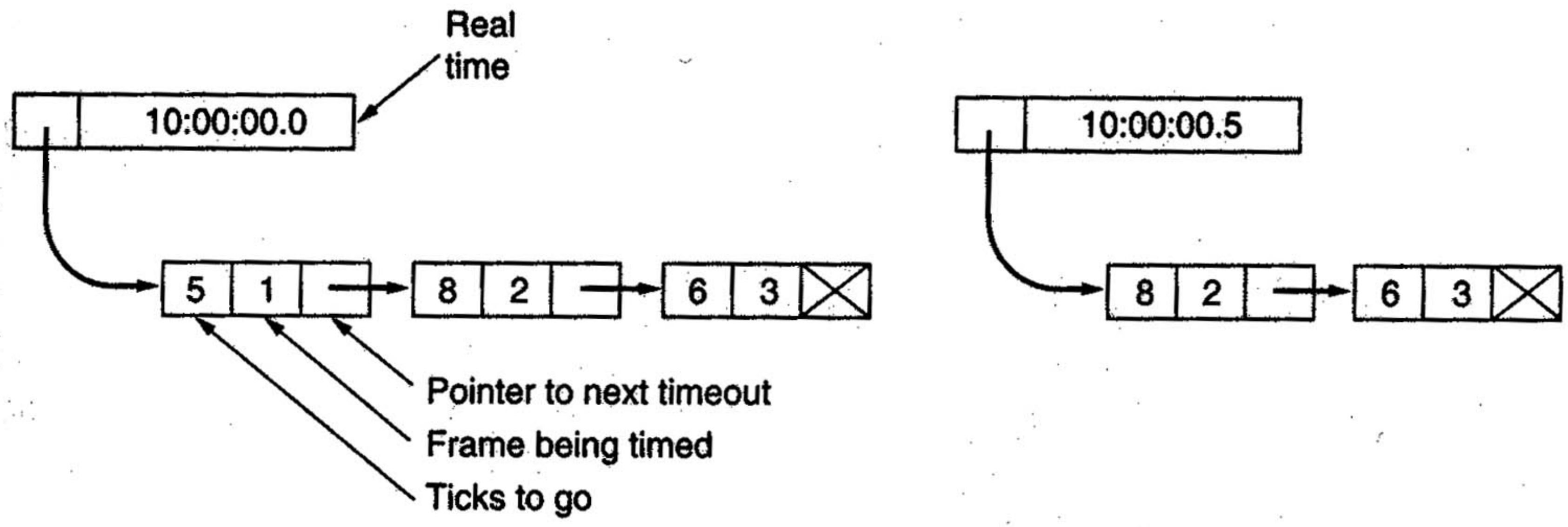
---> Each frame times out independent of all the other ones

---> It logically needs multiple timers

Simulation of multiple timers in software using a single hardware clock

The pending timeouts form a linked list

Example:



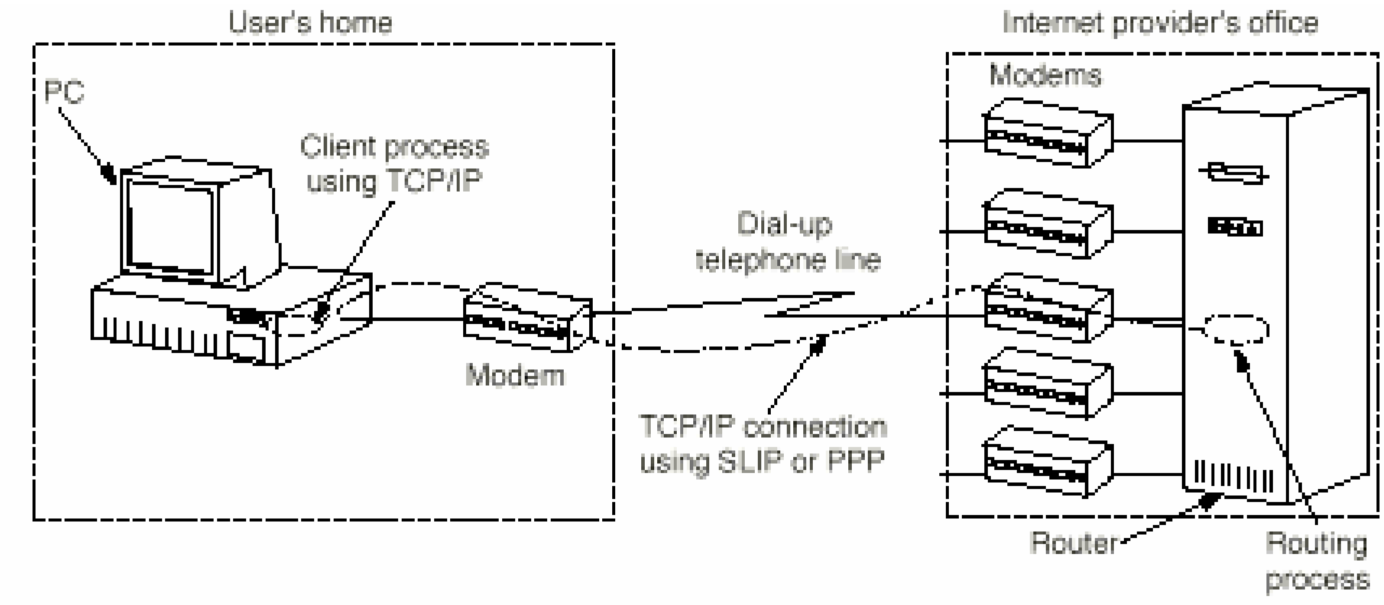
(a) (b)

Data Link Layer(19)

Internet protocols for the Data Link Layer (point-to-point):

Two protocols, SLIP and PPP, are widely used in the Internet as point-to-point data link protocols.

Typical application example: A home PC acting as an Internet host



Data Link Layer(14)

The Serial Line Internet Protocol (SLIP):

- Designed in 1984 to connect SUN workstations to the Internet over a dial-up line using a modem.
- It is very simple:
  - sends raw IP packets over the line with a special flag byte at the end for framing
  - uses some form of character stuffing
- Drawbacks:
  - does not do any error detection or correction
  - supports only IP
  - each side must know the other´s IP address a priori
  - does not provide any form of authentication
  - no approved Internet Standard

Data Link Layer(15)

The Point-to-Point Protocol (PPP):

PPP basically provides three things:

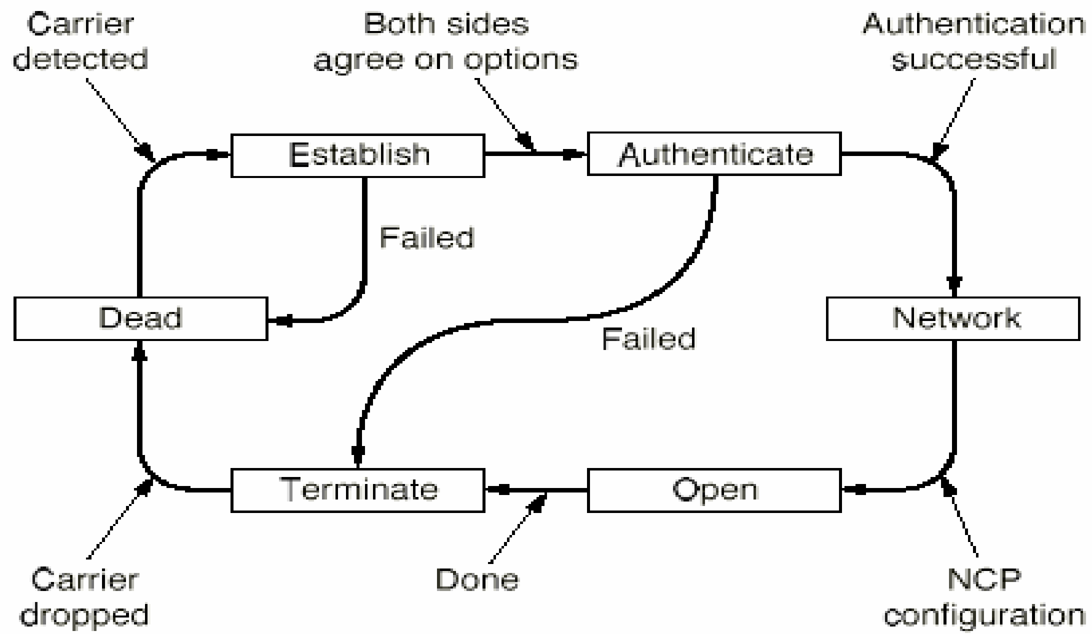
1. A framing method that unambiguously delineates the end of one frame and the start of the next one. The frame format also handles error detection.

2. A link control protocol for bringing lines up, testing them, negotiating options, and bringing them down again gracefully when they are no longer needed. This protocol is called **LCP** (**Link Control Protocol**).

3. A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used. The method chosen is to have a different **NCP** (**Network Control Protocol**) for each network layer supported.

Data Link Layer(16)

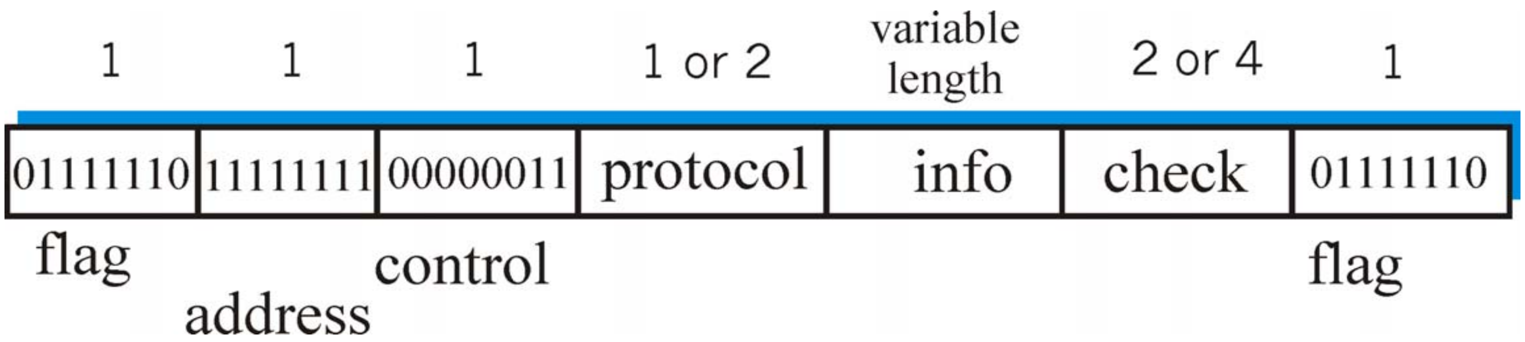
A simplified phase diagram for bringing a line up and down:



The PPP full frame format for unnumbered mode of operation:

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

PPP Data Frame



- Flag: delimiter (framing)
- Address: does nothing (only one option)
- Control: does nothing; in the future possible multiple control fields
- Protocol: upper layer protocol to which frame delivered (eg, PPP-LCP, IP, IPCP, etc)
- info: upper layer data being carried
- check: cyclic redundancy check for error detection