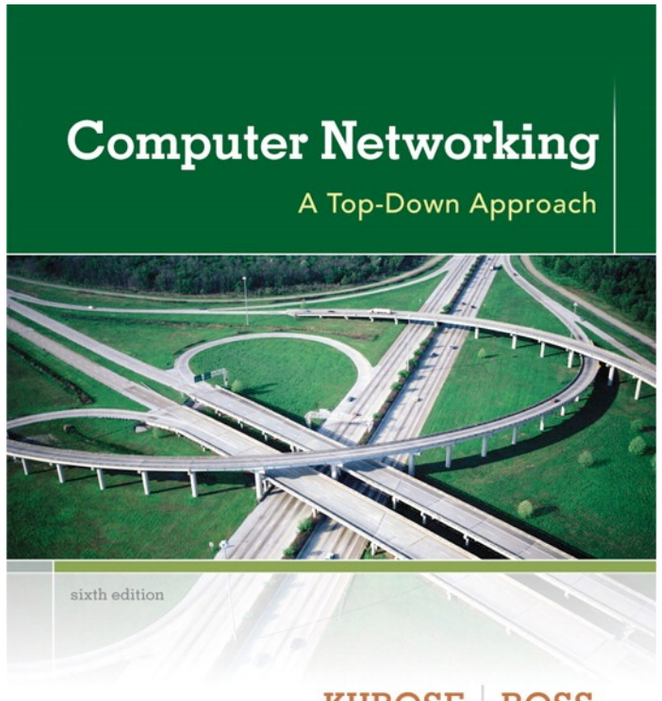
CN-Basic
L23

Reliable Data Transfer
Higher Version

Dr. Ram P Rustagi
rprustagi@ksit.edu.in
http://www.rprustagi.com
https://www.youtube.com/rprustagi

Chapter 3
Transport Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

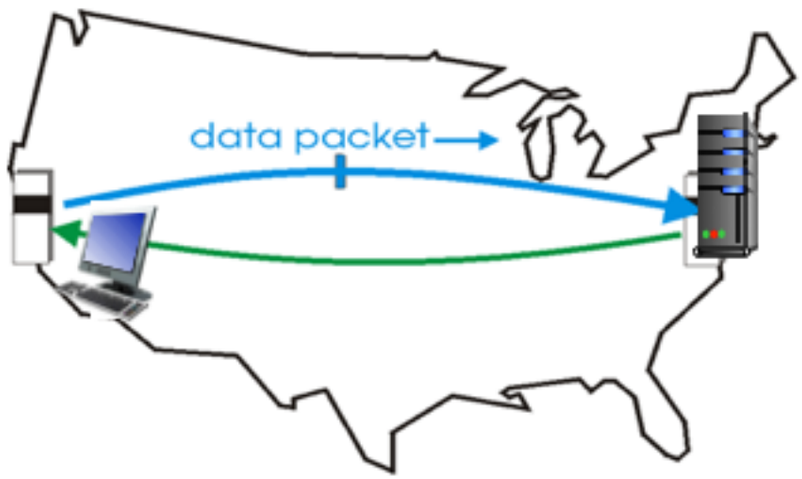
rdt so far

- **rdt 1.0**
  - Fully reliable underlying channel
- **rdt 2.0**
  - No packet loss, but packet corruption can occur
  - Receiver needs to send ACK/NAK
- **rdt 2.1**
  - Introduced seq number to deal with duplicate pkts
- **rdt 2.2**
  - Used ACK instead of NAK
- **rdt 3.0**
  - Underlying channel can also lose packets
  - Timer required for retransmit
  - Efficiency poor for stop-and-wait protocols
- **Next ?**

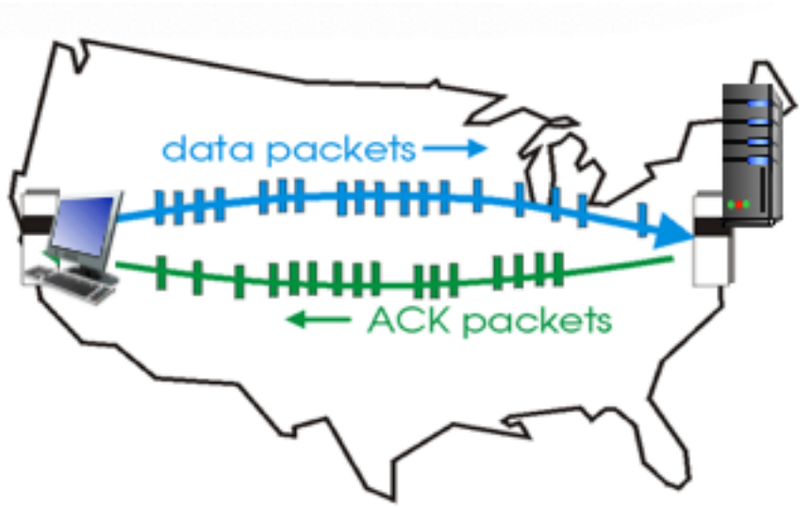
Pipelined protocols

Pipelining: sender sends multiple, “in-flight”, to-be acked

- Range of sequence numbers must be increased
- Buffering at sender and/or receiver
- Increases efficiency



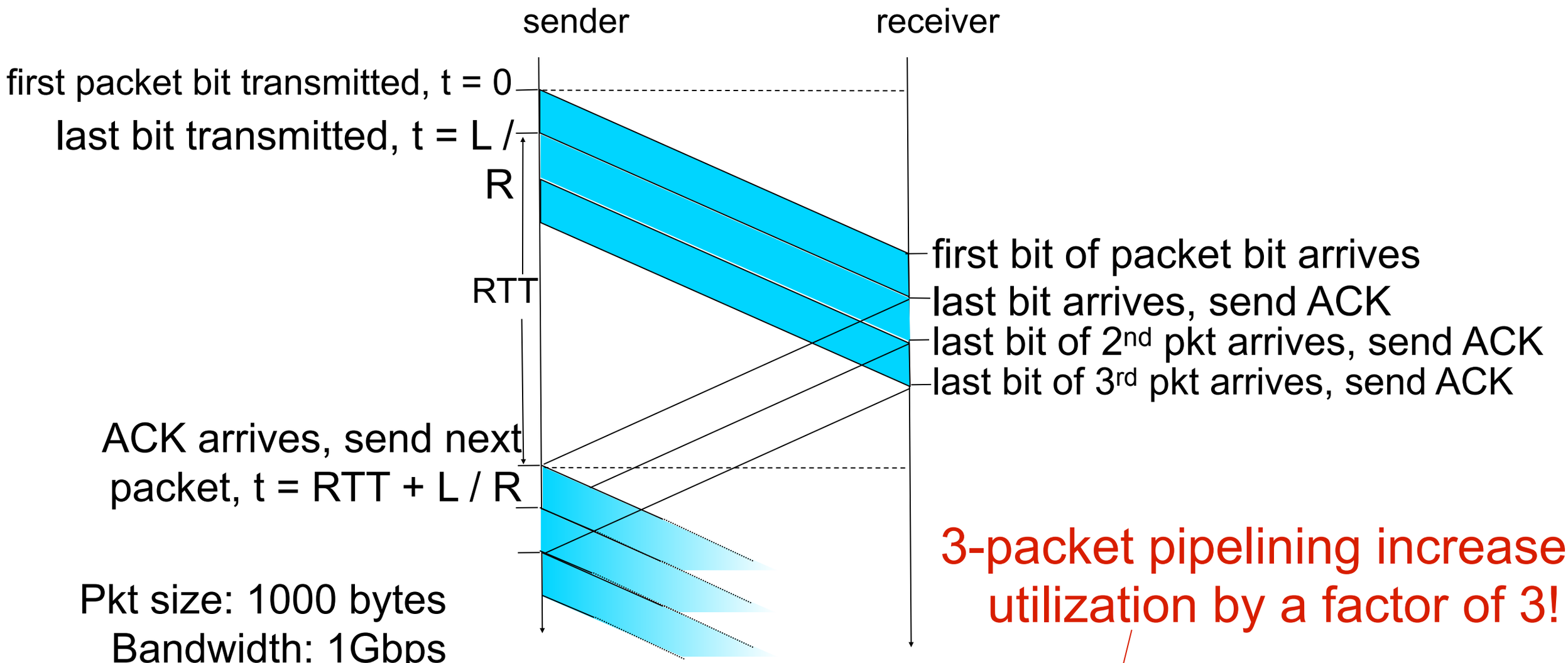
(a) a stop-and-wait protocol in operation



(b) a pipelined protocol in operation

- Two generic forms of pipelined protocols:
  - *Go-Back-N (GBN), Selective Repeat(SR)*

Pipelining: Increased Utilization



Pkt size: 1000 bytes
Bandwidth: 1Gbps
Prop. delay = 15ms

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

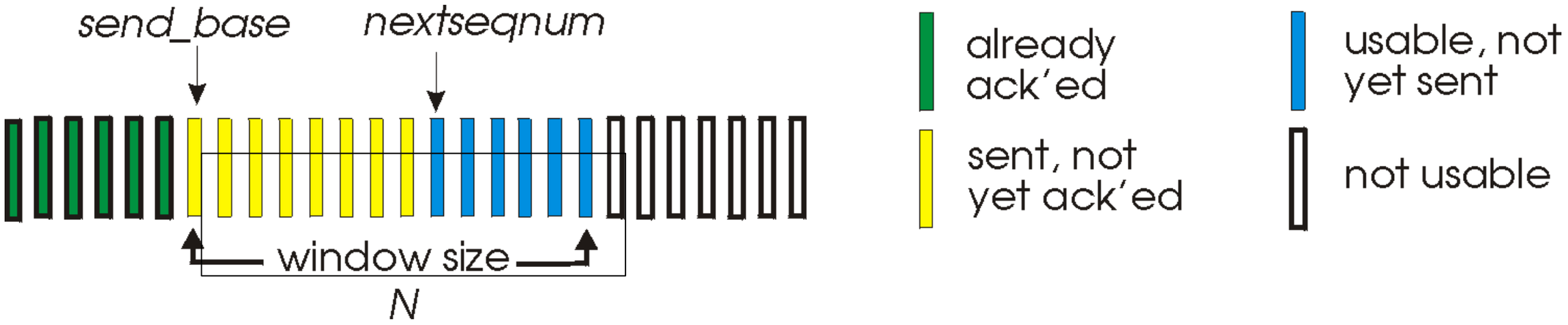
Key point: sender can have up to N unacked pkts in pipeline

- Go-back-N:
- Receiver only sends *cumulative ack*
  - Doesn't ack packet if there's a gap
  - Acks last in-seq pkt received
- Sender has timer for oldest unacked packet
  - When timer expires, retransmit *all* unacked packets

- Selective Repeat:
- Rcvr sends *individual ack* for each packet
  - Buffer requirement?
- 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
- it is a **Sliding Window Protocol**.

Go-Back-N: sender

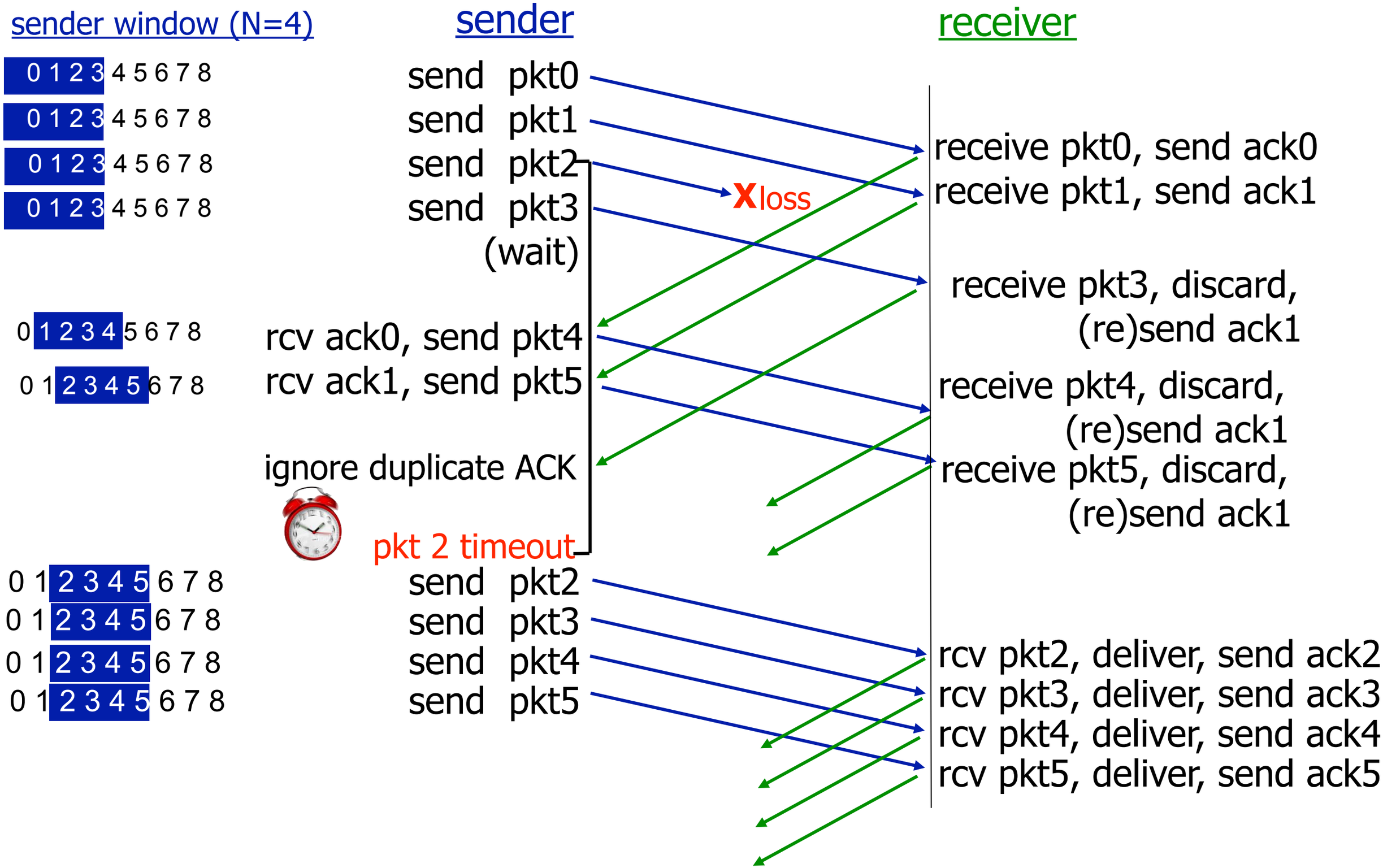
- Why to limit the window size to N
  - What happens if it is made larger (unlimited)?
    - Issues w.r.t flow control (study later)
    - Issues w.r.t. congestion control (study later)
- Given `k` bits for sequence number,
  - Sequence numbers are
    - `0, 1, 2, …, 2^k-1, 0, …`
  - Need to use `modulo 2^k` arithmetic
  - Roll over of seq number has its own set of problems
- GBN Simulation Applet
  - **http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/**
  - **http://computerscience.unicam.it/marcantoni/reti/applet/GoBackProtocol/goback.html**

GBN protocol: example

- Consider the following case
  - Window size = 4
  - Sequence number `0, 1, …, 6` (no roll over)
  - Pkt with seq=`2` (i.e. `3`rd) is lost and hence times out
  - A total of `6` packets are to be transmitted

- Exercise:
  - Draw the timeline of packets and ack communication

GBN in action



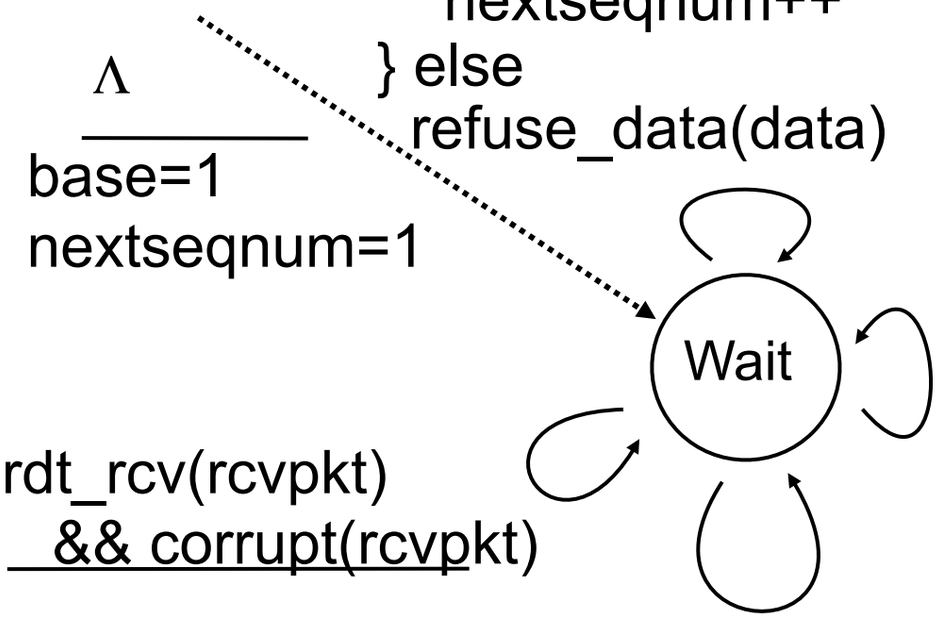
GBN: Sender Extended FSM

rdt_send(data)

```
if (nextseqnum < base+N) { //ensure window is not full
    sndpkt[nextseqnum] = make_pkt(nextseq,data,chksum)
    udt_send(sndpkt[nextseqnum])
    if (base == nextseqnum)
       start_timer
    nextseqnum++
} else
    refuse_data(data)
```

Three event types

- invocation from above
- timeout
- receipt of an ack
  - corrupt or uncorrupt

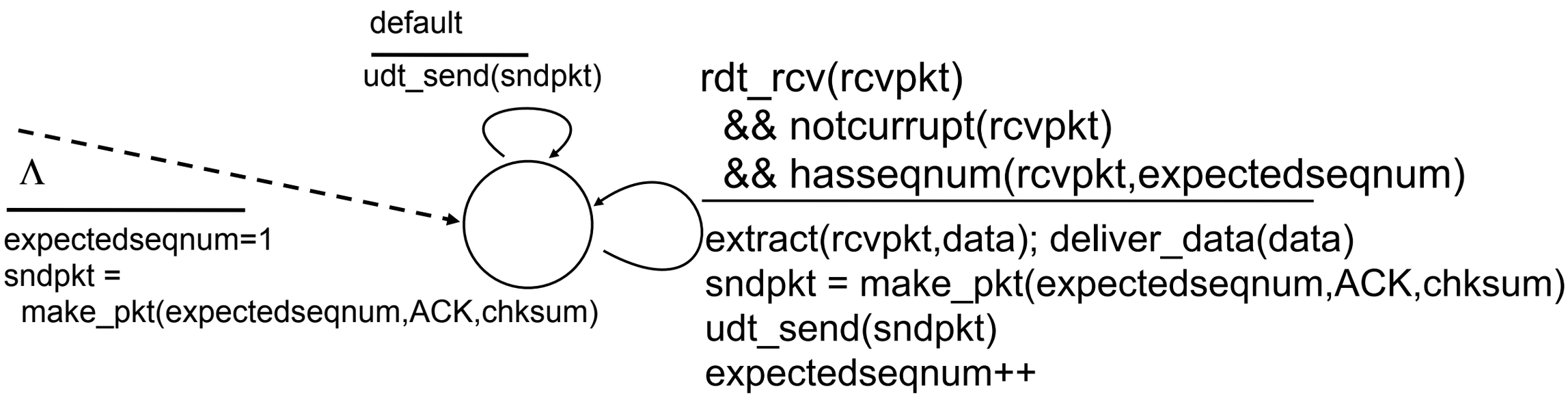


timeout // gives the protocol name GBN

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

```
base = getacknum(rcvpkt)+1 //cumulative ack mechanism
If (base == nextseqnum)
    stop_timer
else
    start_timer
```

GBN: receiver extended FSM



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!*
    - *Does buffering help? Will sender anyway resend it?*
  - Re-ACK pkt with highest in-order seq #

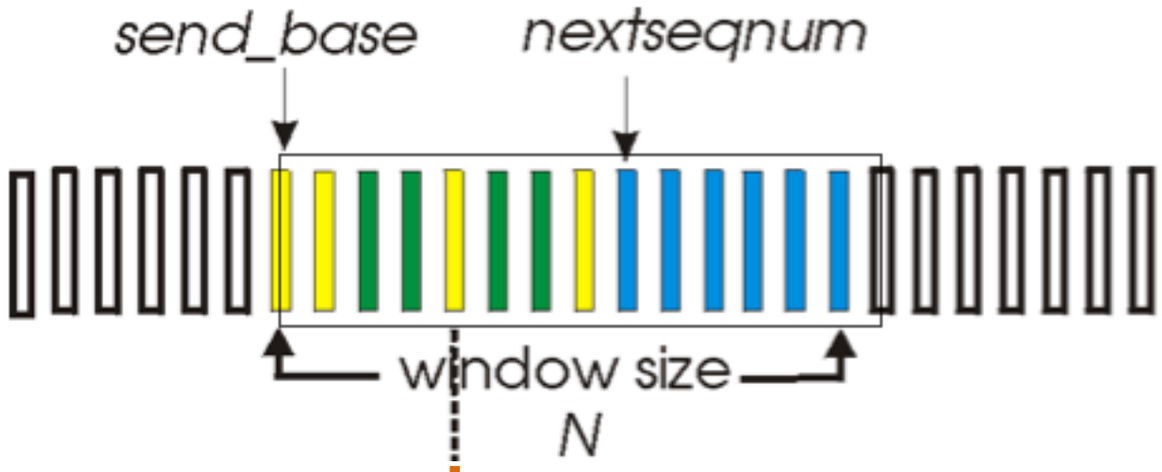
GBN in action

- Benefits over stop-n-wait
  - Fills the pipeline, efficient channel utilization
- Issues
  - Performance problems if BW-delay product is large
  - A single packet error causes large num of retransmissions
    - Most of them unnecessary
  - With increasing error rates
    - Channel is filled with retransmissions
- Solution: **Selective Repeat**

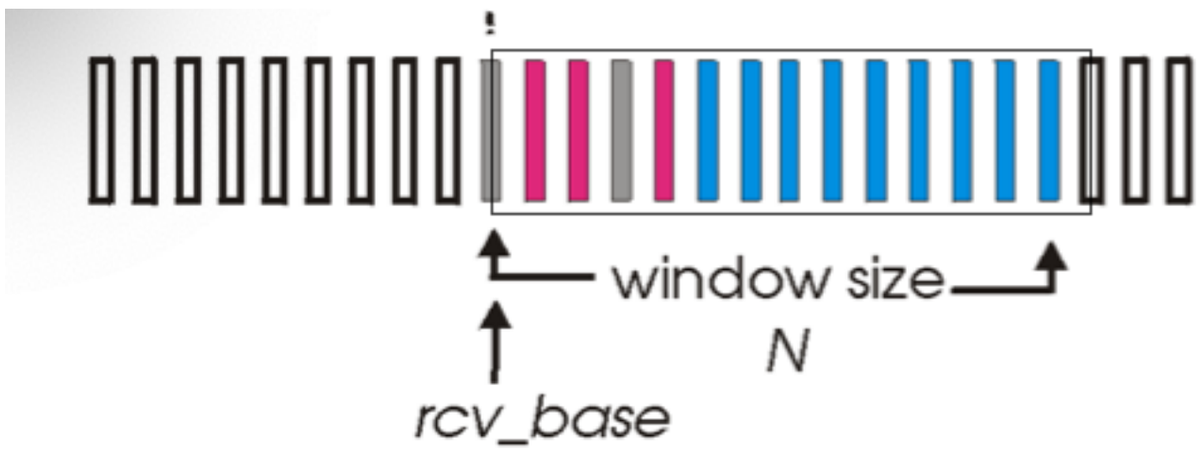
Selective Repeat

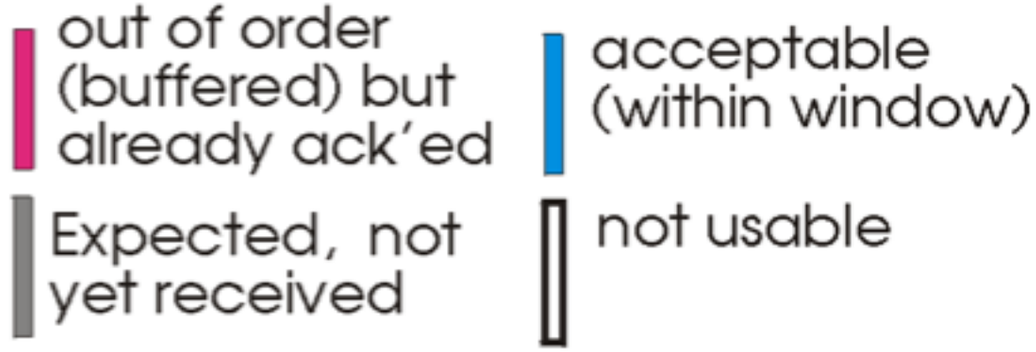
- Receiver *individually* acknowledges all correctly received pkts
  - Buffers pkts, as needed, for eventual in-order delivery to upper layer
  - Acks need not be in order
- Sender only resends pkts for which ACK not received
  - Sender timer for each unACKed pkt
    - More complexity at sender
- Sender window
  - *N* consecutive seq #'s
  - Limits seq #s of sent, unACKed pkts

Selective repeat: sender, receiver windows



Sender view of sequence number





Receiver view of sequence number

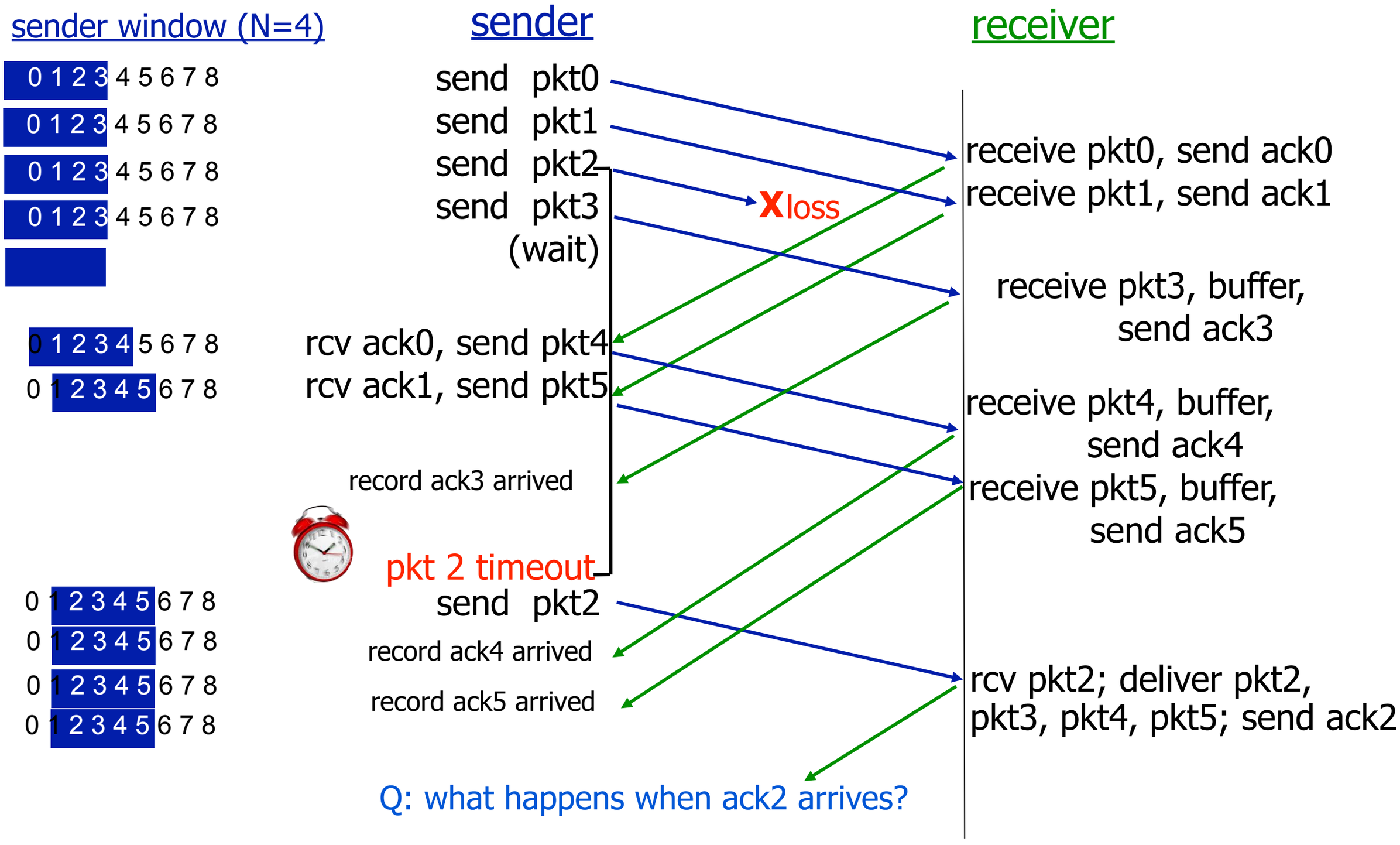
Selective repeat in action

- SR Simulation Applet
  - http://computerscience.unicam.it/marcantoni/reti/applet/SelectiveRepeatProtocol/selRepProt.html
- Observation
  - Receiver re-acknowledges packets even below its current window base
    - Is it really needed?
  - Sender and receiver windows typically do not coincide
    - Causes issues w.r.t. finite range of seq numbers

Selective repeat in action

- Consider the following scenario
  - Window size of 4
  - Seq number starting from 0, 1, …
    - `pkt0, pkt1, pkt2, pkt3, pkt4, pkt5, …`
  - `pkt2` is lost
  - Draw the Timeline Sequence diagram for SR protocol

Selective repeat in action



Selective repeat(actions by sender/receiver)

sender

- Data from above:
  - If next available seq # in window, send pkt, start timer
- 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 #

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

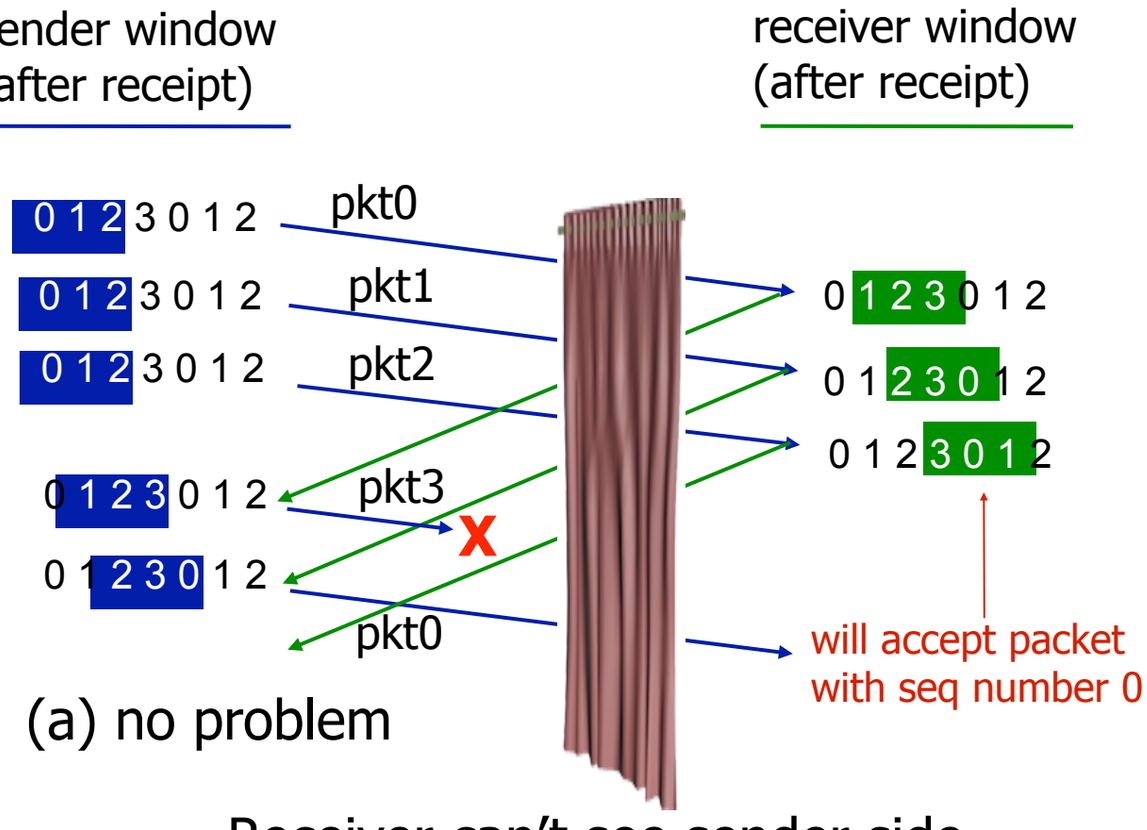
- Home Exercise
  - Draw the FSM for Sender and receiver

- Issues with SR
  - When seq numbers range is close to window size?
    - Window size 3,
    - Sequence number (only 4 values) 0, 1, 2, 3.

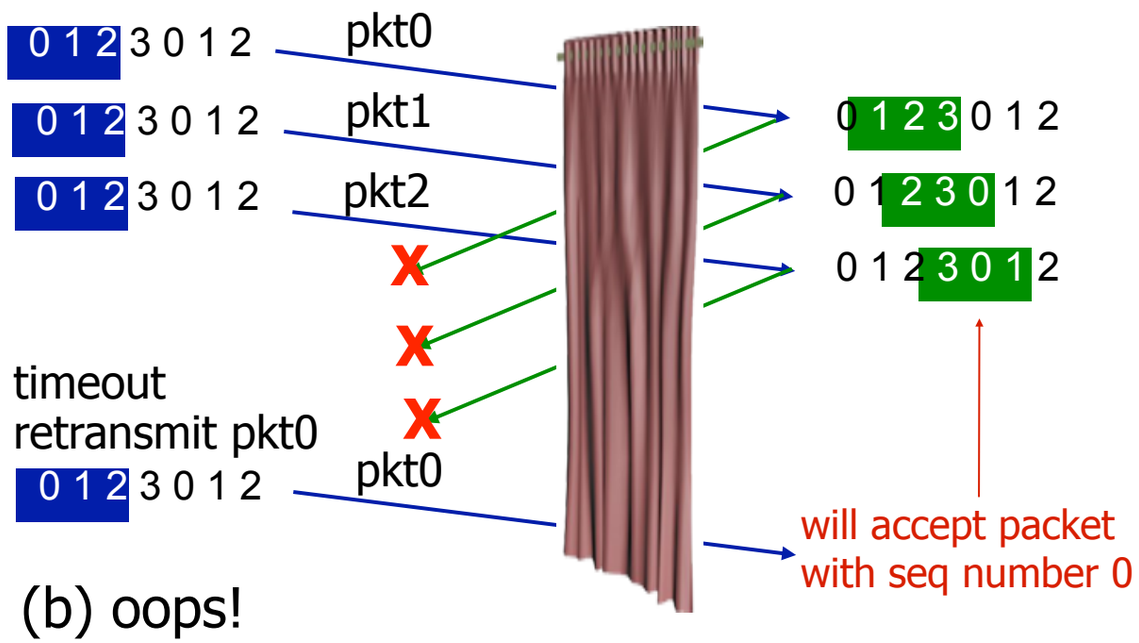
Selective Repeat: dilemma

- 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)?



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



Selective repeat dilemma

- Figurative curtain between sender and receiver
  - Receiver can't see action taken by sender
  - From receiver perspective
    - Two scenario (prev slide) are identical
    - No way to distinguish between
      - Retransmission of pkt `#0` with original pkt `#5`
- Issue:
  - Window size of `1` less than size of seq number
  - How small the window size should be?
    - Less than N/2

Summary of **rdt**

- Numerous mechanism for **rdt**(reliable data transfer)
  - Checksum, timer, seq number, ACK, NAK, window size, pipelining
- Mechanisms were incrementally added
  - To address increasingly complex (realistic) problems
- Underlying assumption
  - Channel sends packets in order, does not reorder them
  - Not true in real life
    - Multiple paths between sender and receiver
  - Impact of packet re-order
    - Pkt with old seq num x (or ack) can re-appear
      - Though neither side contains x
    - How to address such duplicate packets
      - Sender should use a seq #, only when sure no such pkts exists in the network
      - Assumes pkt has a lifetime (TCP assumes it to be 3 minutes)