# Network Programming

Dr. Thaier Hayajneh

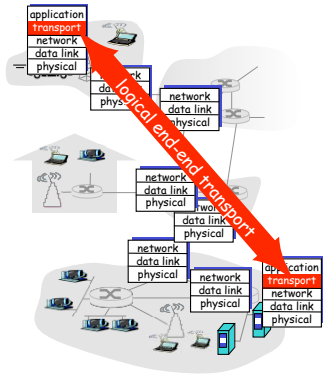
Computer Engineering Department

## *Principles of Reliable data transfer*



## Internet transport-layer protocols

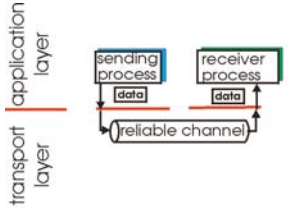
- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees






## Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!



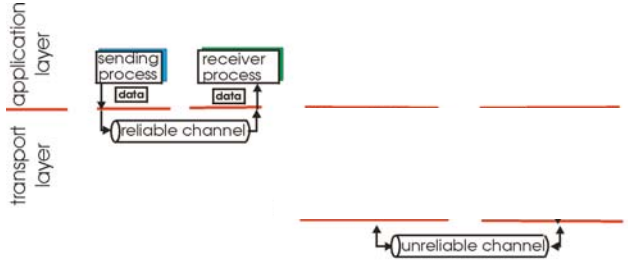

(a) provided service

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)



## Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

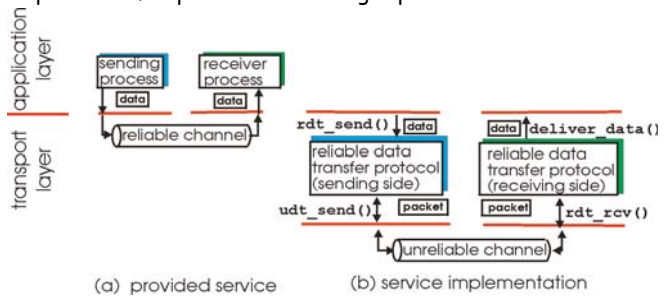



(a) provided service (b) service implementation

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

## Principles of Reliable data transfer

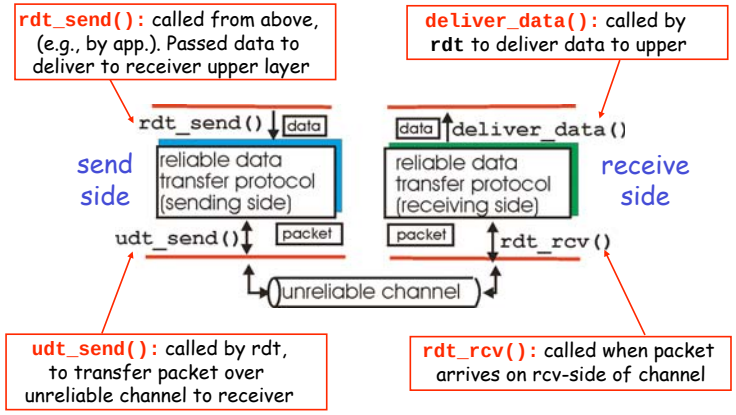
- important in app., transport, link layers
- top-10 list of important networking topics!




- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)



## Reliable data transfer: getting started

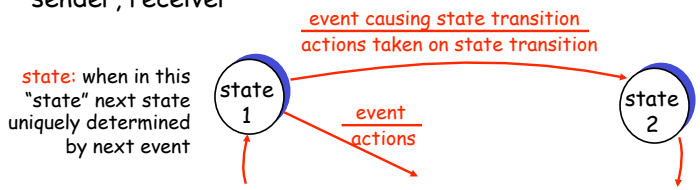





## Reliable data transfer: getting started

We'll:

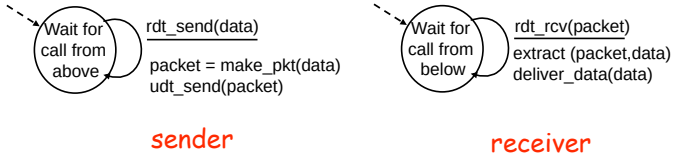
- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver






## Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel

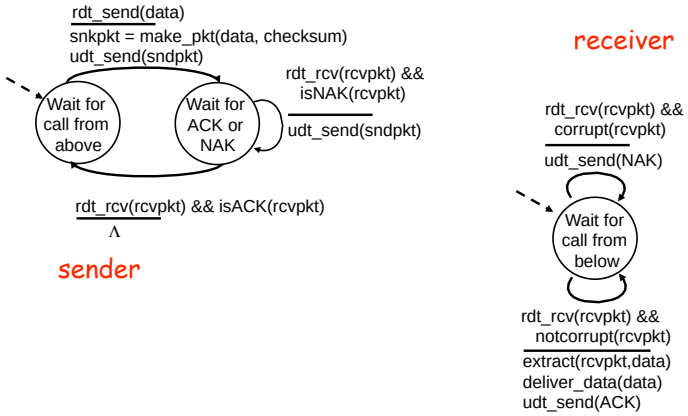


## Rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- new mechanisms in **`rdt2.0`** (beyond **`rdt1.0`**):
  - error detection
  - receiver feedback: control msgs (ACK,NAK) rcvr->sender
  - Retransmission: packets received in error by receiver will be retransmitted by the sender

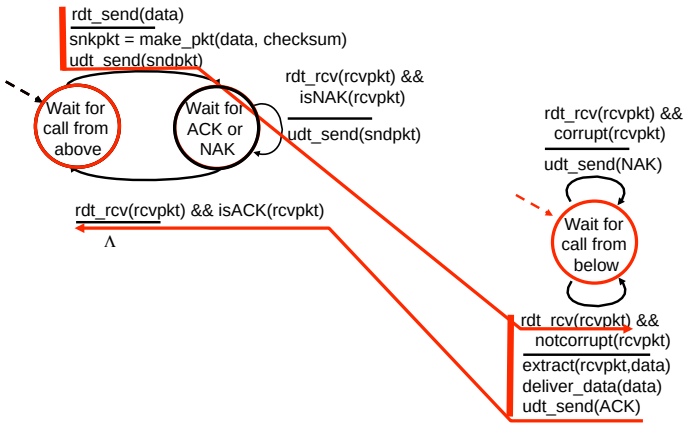


## rdt2.0: FSM specification



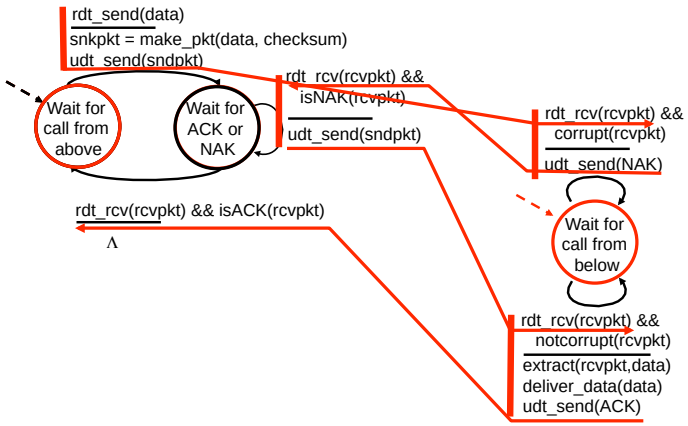



## rdt2.0: operation with no errors






## rdt2.0: error scenario



# 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:

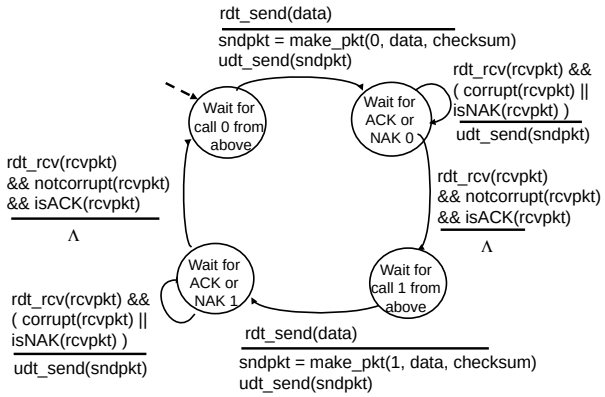
- sender retransmits current pkt if ACK/NAK garbled
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait

Sender sends one packet, then waits for receiver response

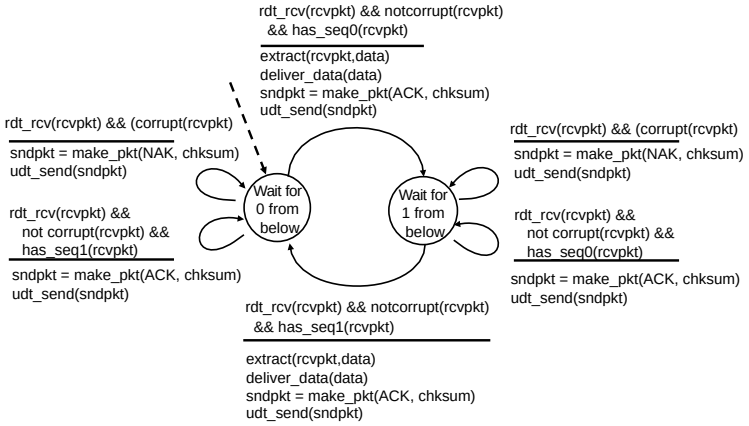


# rdt2.1: sender, handles garbled ACK/NAKs






# rdt2.1: receiver, handles garbled ACK/NAKs






# rdt2.1: discussion

Sender:

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

Receiver:

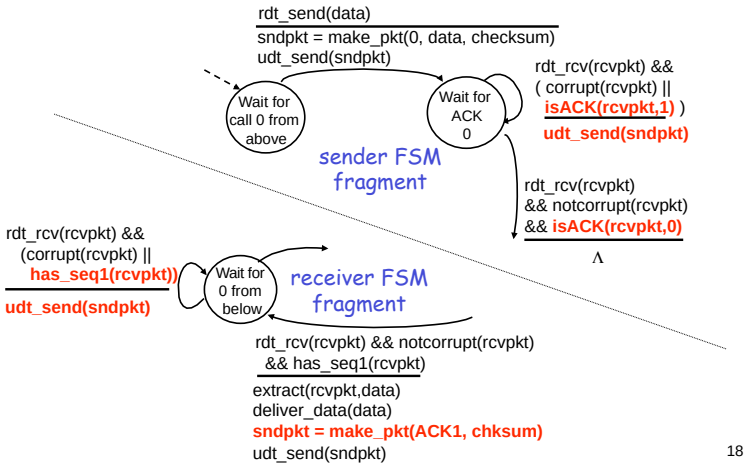
- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/ NAK received OK at sender

## 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





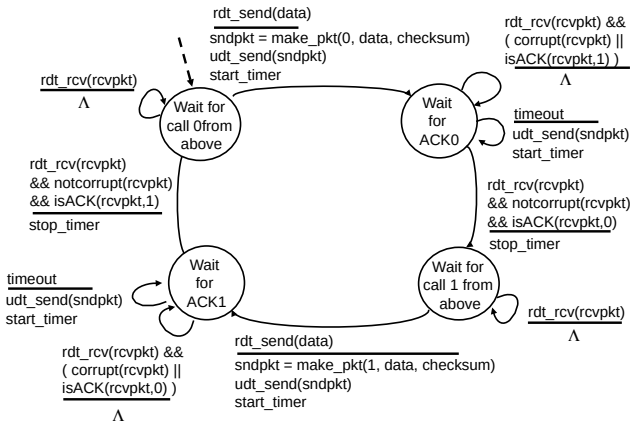

## rdt3.0: channels with errors *and* loss

New assumption: underlying channel can also lose packets (data or ACKs)
- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

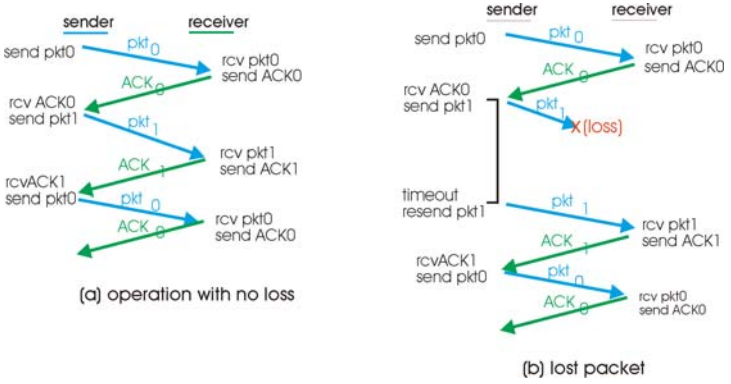
Approach: sender waits "reasonable" amount of time for ACK
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer



## rdt3.0 sender

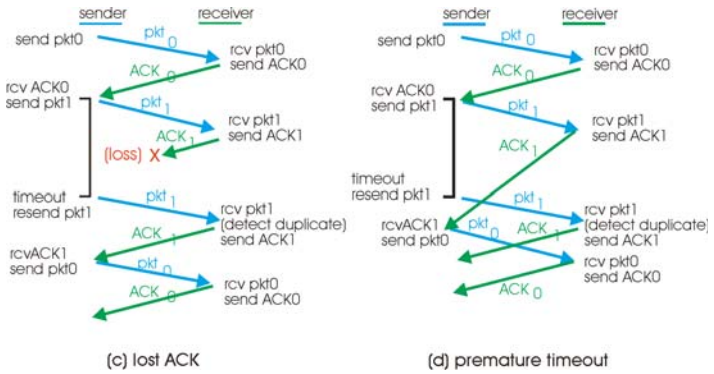


## rdt3.0 in action






## rdt3.0 in action






## Performance of rdt3.0

- rdt3.0 works, but performance stinks
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\,\text{bps}} = 8\,\text{microseconds}$$

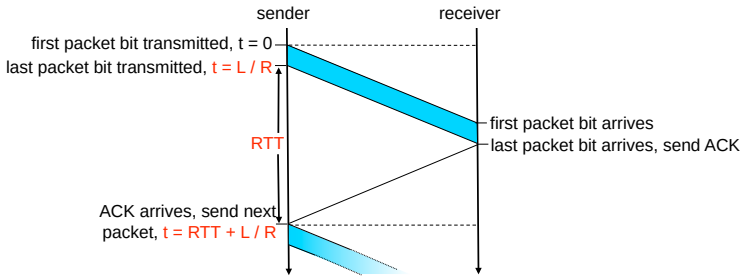
  - $U_{sender}$: utilization – fraction of time sender busy sending

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

  - 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

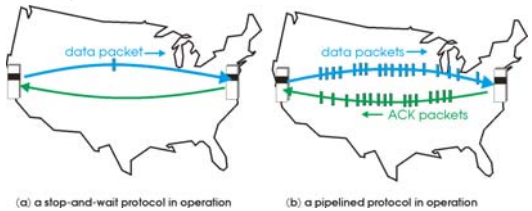



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

# Pipelined protocols

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

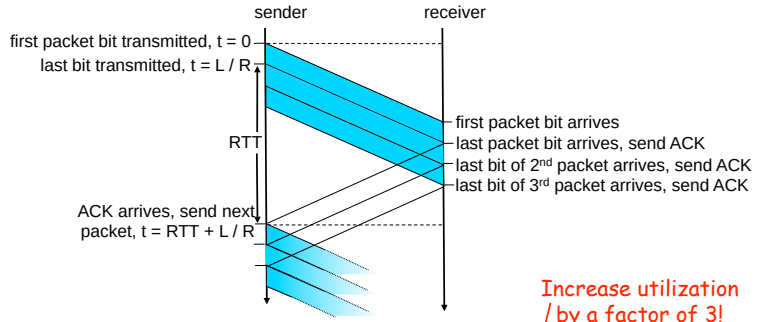
- range of sequence numbers must be increased
- buffering at sender and/or receiver




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



# Pipelining: increased utilization




Increase utilization by a factor of 3!

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



# Pipelining Protocols

Go-back-N: big picture:

- Sender can have up to N unacked packets in pipeline
- Rcvr only sends cumulative acks
  - Doesn't ack packet if there's a gap
- Sender has timer for oldest unacked packet
  - If timer expires, retransmit all unacked packets

Selective Repeat: big pic

- Sender can have up to N unacked packets in pipeline
- Rcvr acks individual packets
- Sender maintains timer for each unacked packet
  - When timer expires, retransmit only unack packet



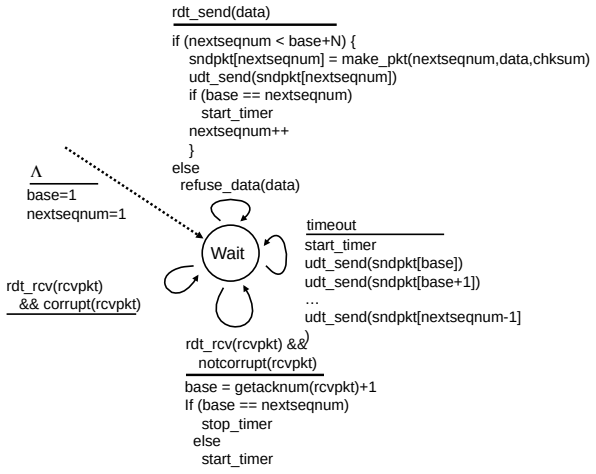
# Go-Back-N

Sender:

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

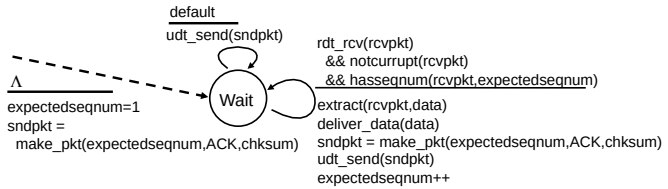

- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

## GBN: sender extended FSM






## GBN: receiver extended FSM



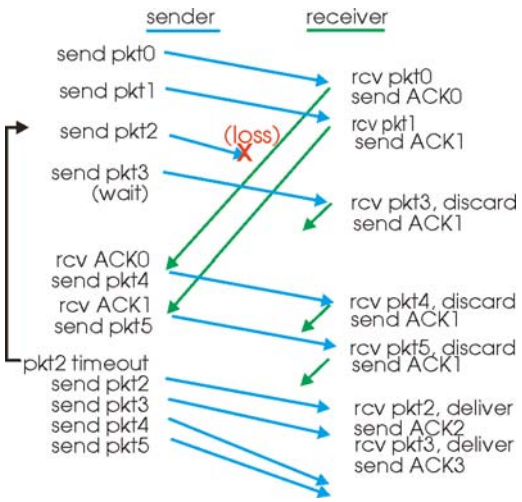

ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

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



## GBN in action

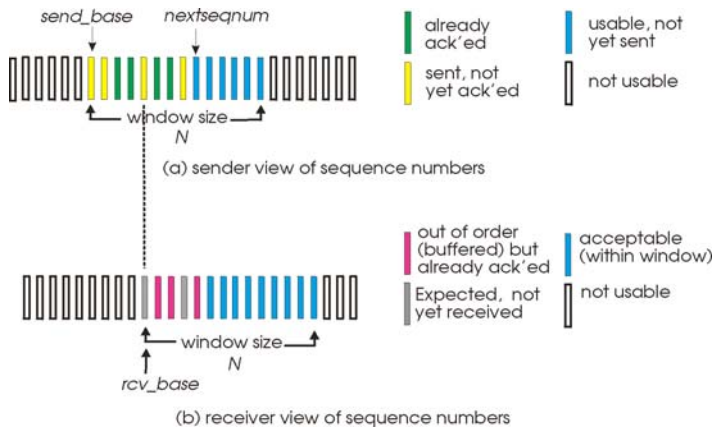





## Selective Repeat

- 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
  - again limits seq #s of sent, unACKed pkts

## Selective repeat: sender, receiver windows






## Selective repeat

**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 #

**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]

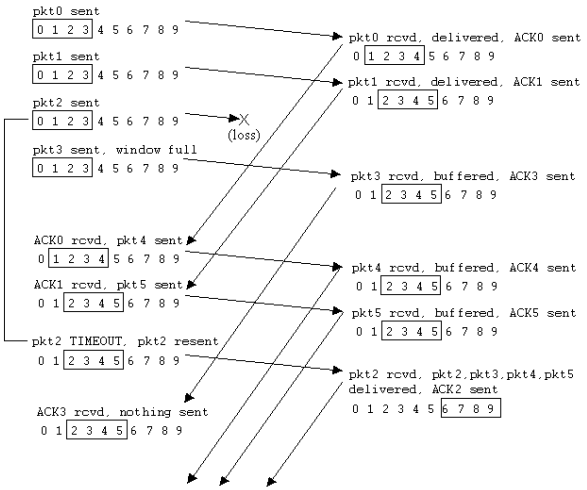
- ACK(n)

otherwise:

- ignore



## Selective repeat in action





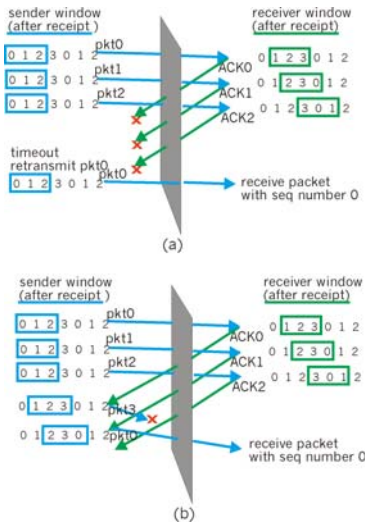

## Selective repeat: dilemma

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?



- Project Phase 2 is available at:

http://www.eis.hu.edu.jo/ACUploads/10799/Project_P2_F2011.pdf

37