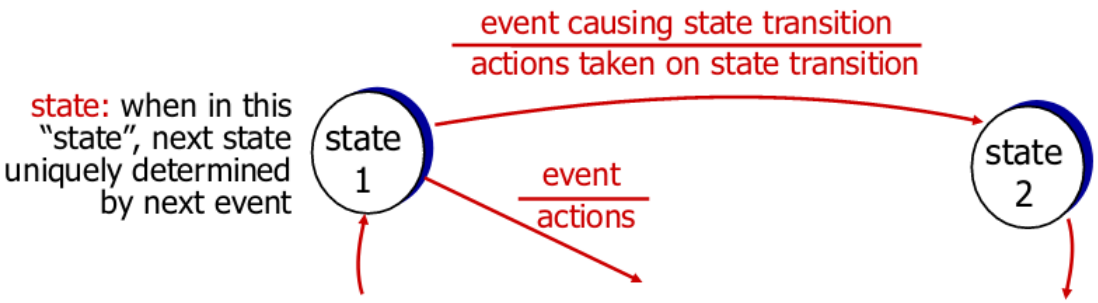
Transport Layer II

1 Reliable data transfer

We will:

- 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



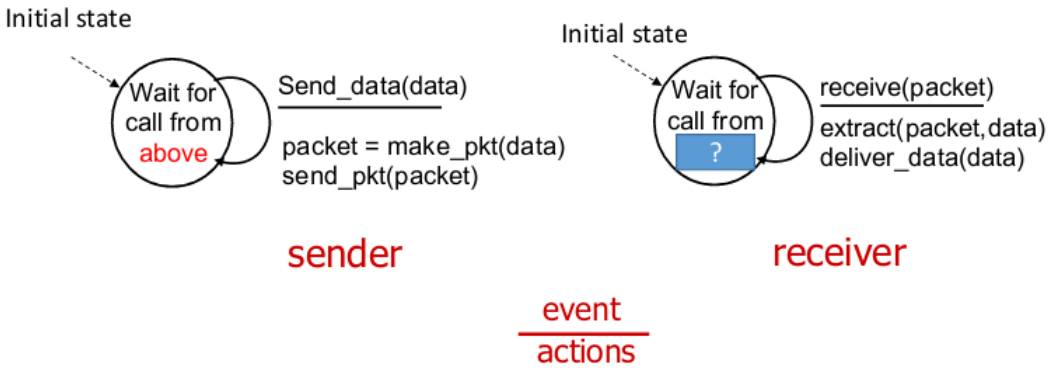
1.1 rdt1.0: reliable transfer over a reliable channel

Underlying channel perfectly reliable:

- No bit errors
- No loss of packets

Separate FSMs for sender, receiver:

- Sender sends data into underlying channel
- Receiver reads data from underlying channel



1.2 rdt2.0

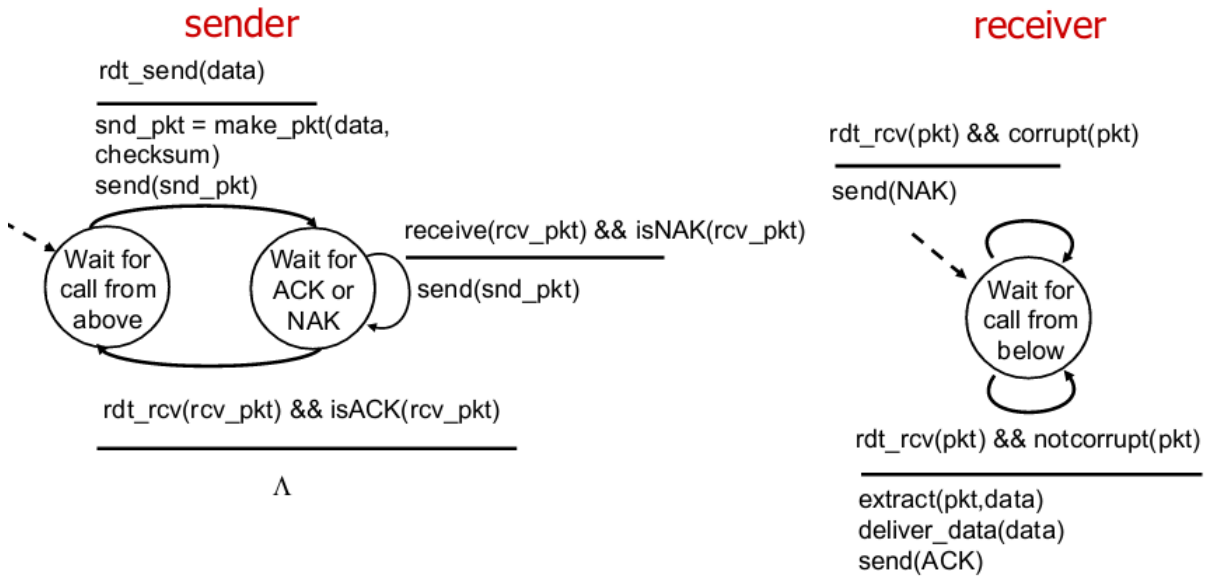
1.2.1 Channel with bit errors

- 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

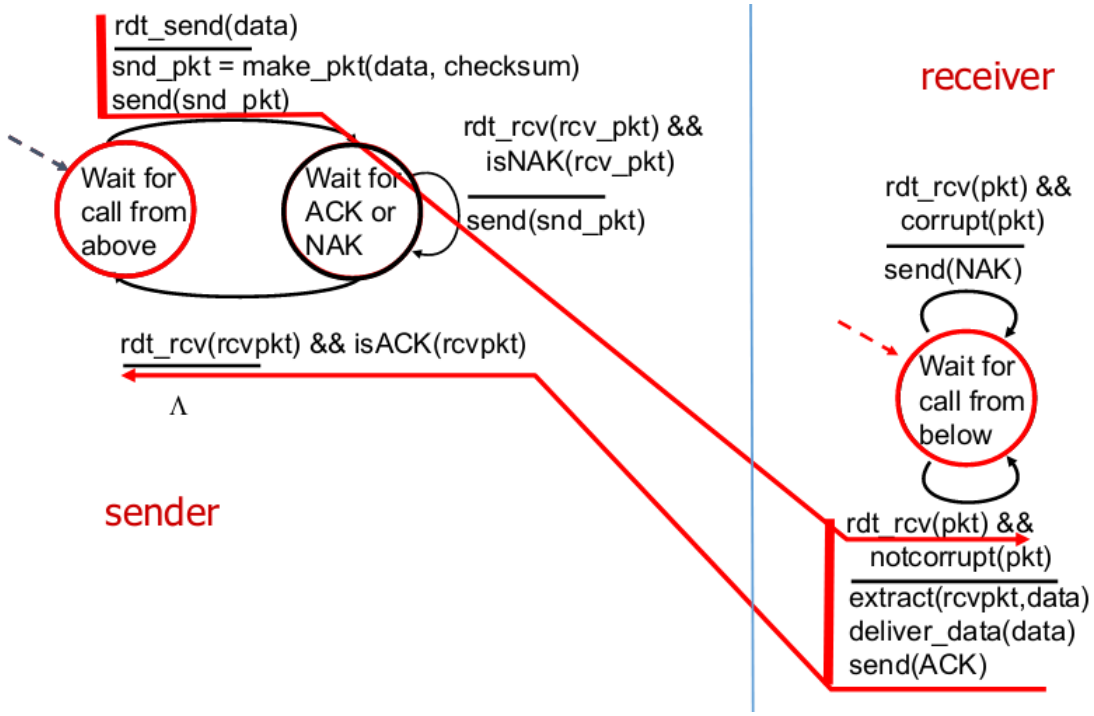
  - Sender retransmits pkt on receipt of NAK
  - Using ACKs and NAKs is known as ARQ (Automatic Repeat reQuest) protocols
    * Error detection. Sender embeds extra bits in packets
    * Feedback. Receiver provide sender with feedback
    * Retransmission. Retransmit erroneous packets

- New mechanisms in `rdt2.0` (beyond rdt1.0)
  - Error detection
  - Feedback: control msgs (ACK (1), NAK(0)) from receiver to sender

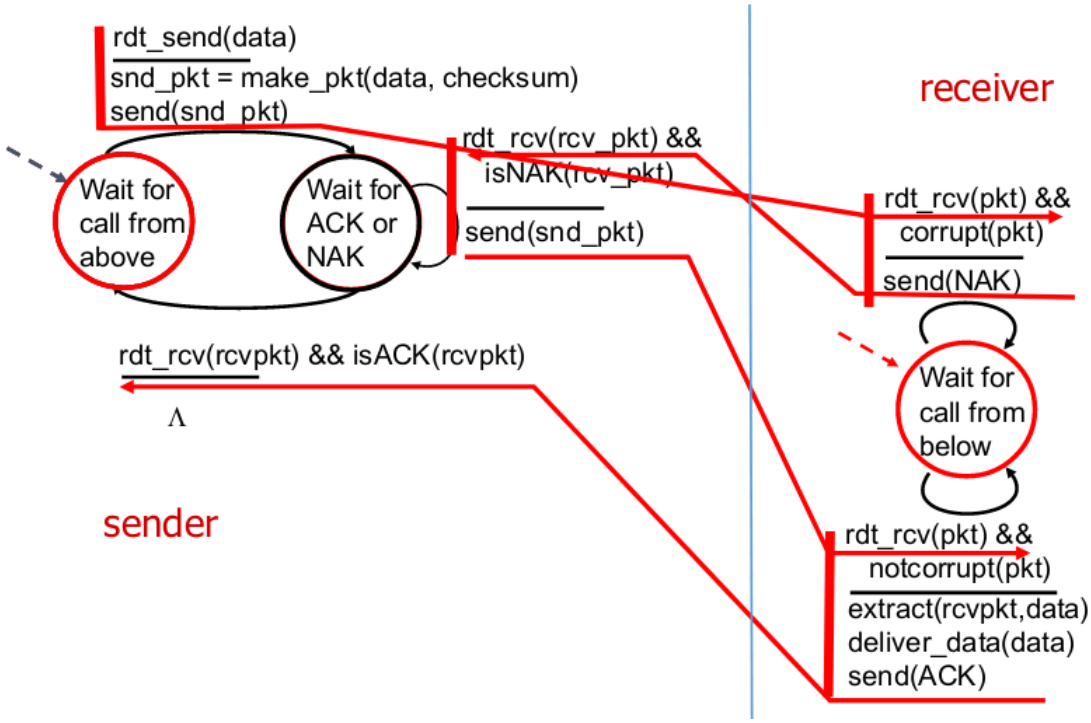
1.2.2 FSM specification



1.2.3 Operation with no errors



1.2.4 Error scenario



1.2.5 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 the receiver response

1.3 rdt3.0

1.3.1 Channels with errors and loss

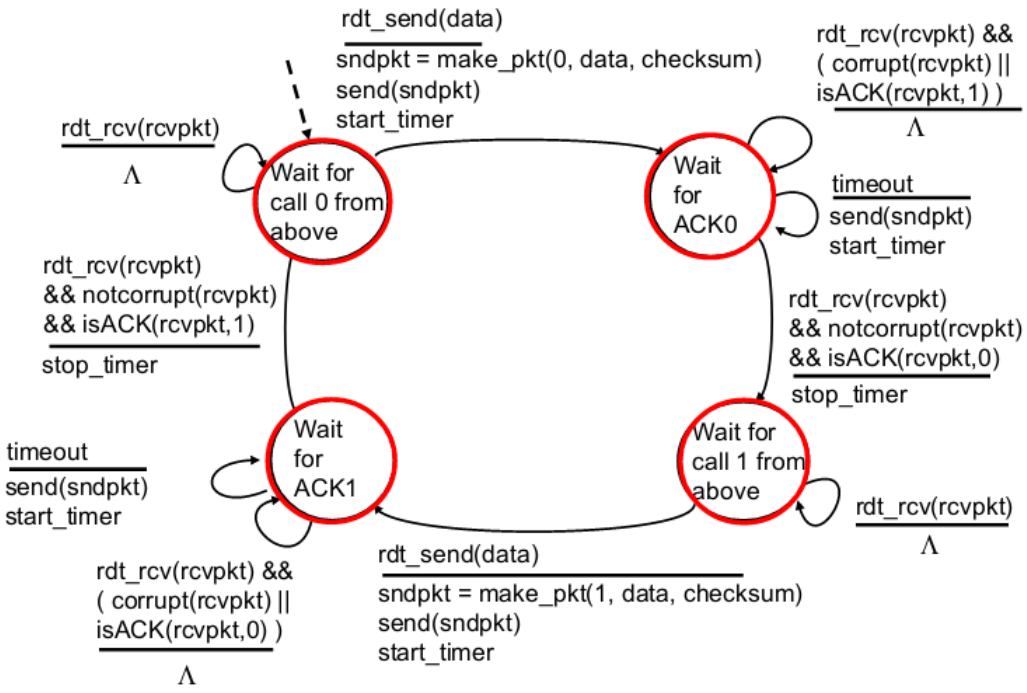
New assumption:

- Underlying channel can also lose packets (data, ACKs)
  - Checksum, seq. #, ACKs, retransmissions will be of help, but not enough

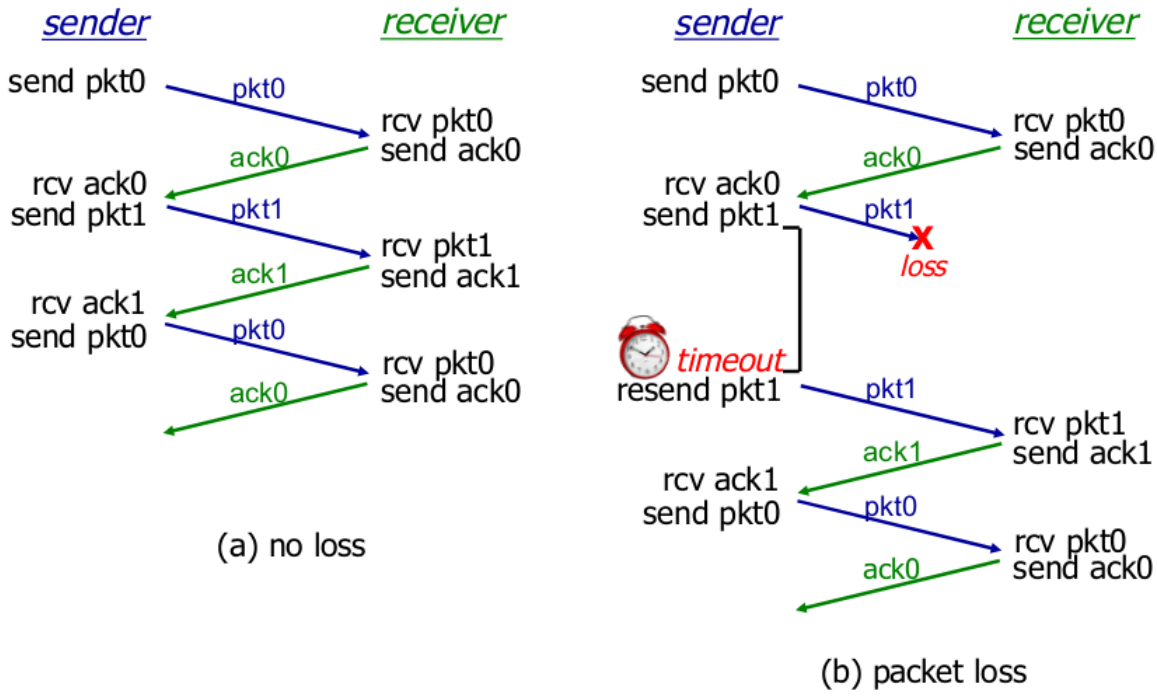
Approach:

- Sender waits "reasonable" amount of time for ACK
  - Retransmits if no ACK received in this time
  - If no 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

1.3.2 Sender

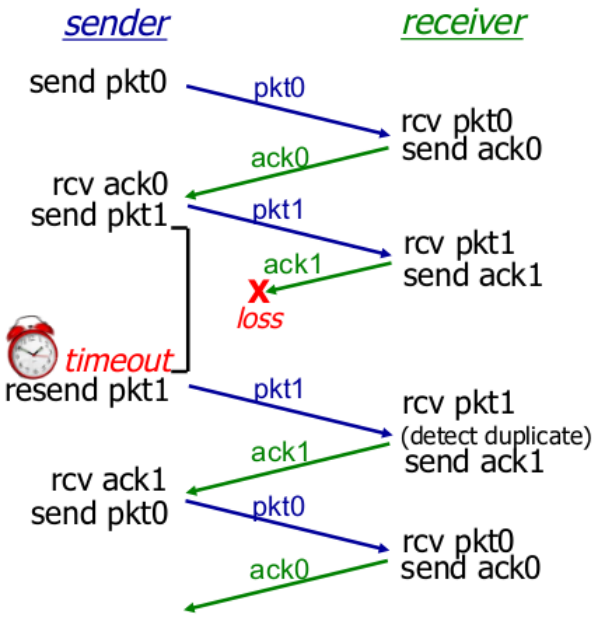


1.3.3 In action

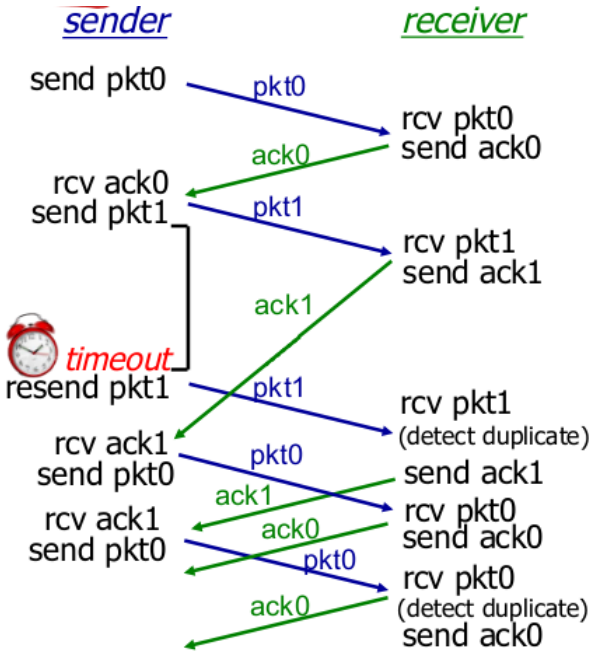


Note: **alternating-bit protocol**: packet sequence numbers alternate between 0 and 1.

14

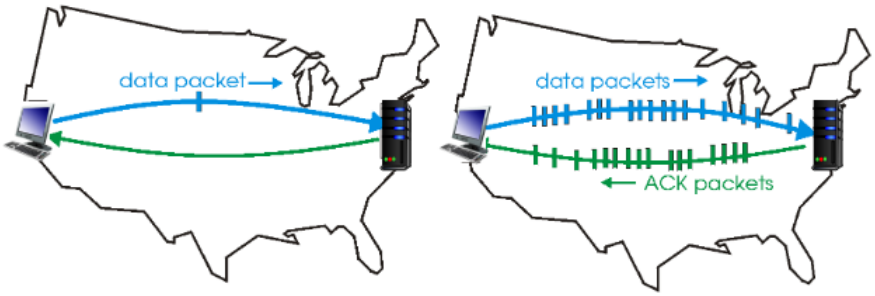


(c) ACK loss



(d) premature timeout/ delayed ACK

2 Pipelined protocols



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

Pipelining has the following consequences for reliable data transfer protocols:

- Range of sequence numbers must be increased
  - Unique sequence number and there may be multiple, in-transit, unacknowledged packets
- Multiple packet buffering at sender and/or receiver
  - Sender buffers packets that have been transmitted but not yet acknowledged
  - Buffering of correctly received packets
- Range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets

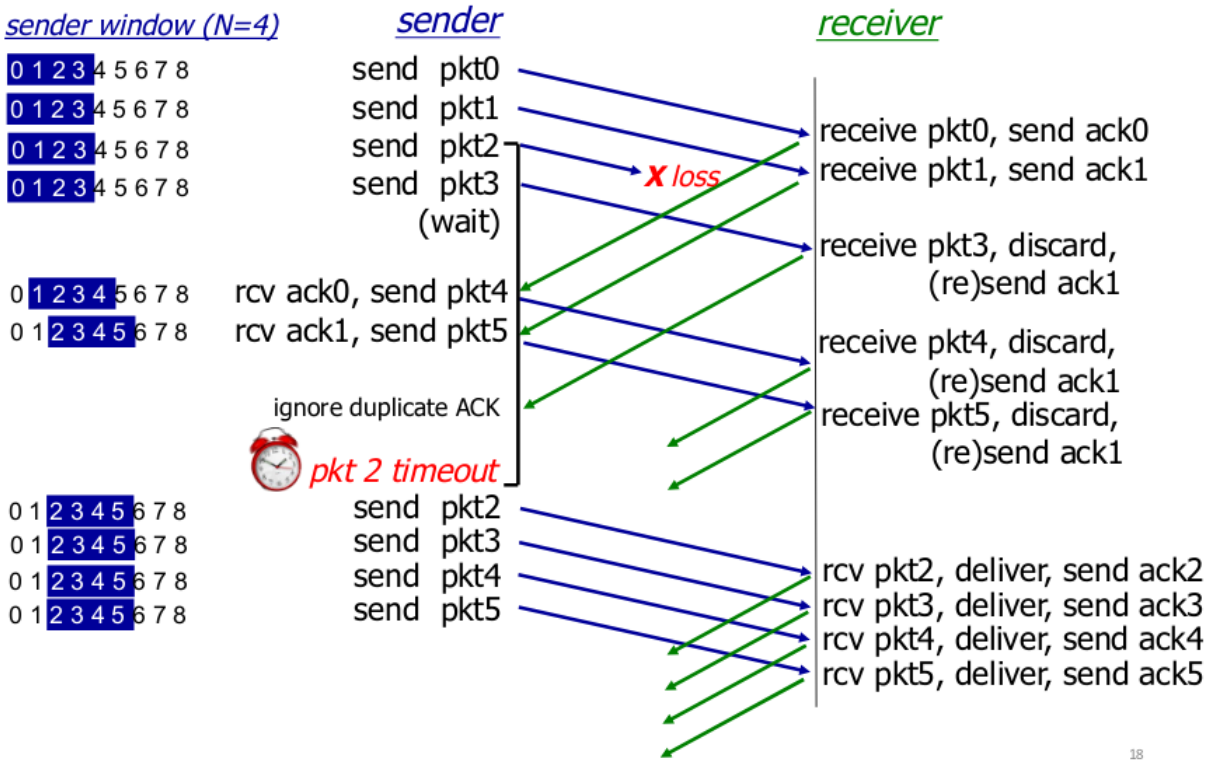
Two generic forms of pipelined protocols:

- Go-back-N
  - Sender can send multiple packets without waiting for ACK
  - Sender can have up to N unacked packets in pipeline
  - Receiver only sends cumulative ack, doesn't ack packet if there's a gap

  - Sender has timer for oldest unacked packet, when timer expired, retransmit all unacked packets

- Selective repeat:
  - Sender can have up to N unacked packets in pipeline
  - Receiver sends individual ack for each packet
  - Sender maintains timer for each unacked packet, when timer expires, retransmit only that unacked packet

3 GBN in action



4 Selective repeat

- Receiver individually acknowledges all correctly receives 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
  - Limits seq #s of sent, unACKed pkts

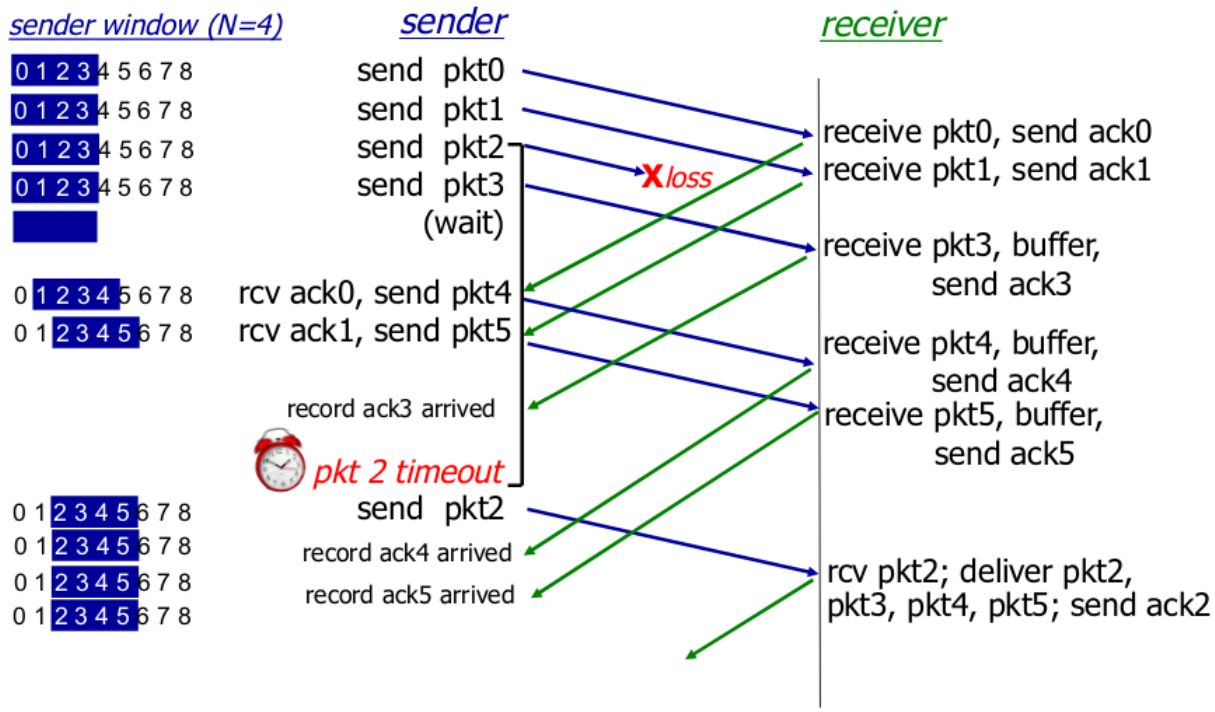
Sender:

- Data from above:
  - If next available seq # in window, send pkt
- timeout(n)
  - Resend pkt n, restart timer
- 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

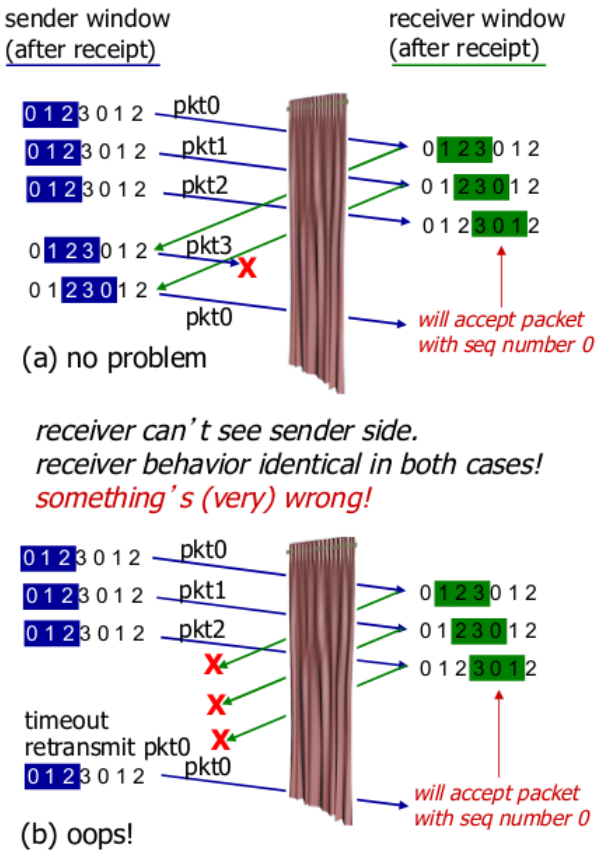
4.1 Selective repeat in action



4.2 Selective repeat dilemma

Example:

- Finite range of seq # s: 0,1,2,3
- Window size=3
  - Receiver sees no difference in two scenarios
  - Duplicate data accepted as new in (b)


sender window
(after receipt)
receiver window
(after receipt)
0 1 2 3 0 1 2
pkt0
pkt1
pkt2
pkt3
pkt0
will accept packet
with seq number 0
(a) no problem
receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!
timeout
retransmit pkt0
(b) oops!