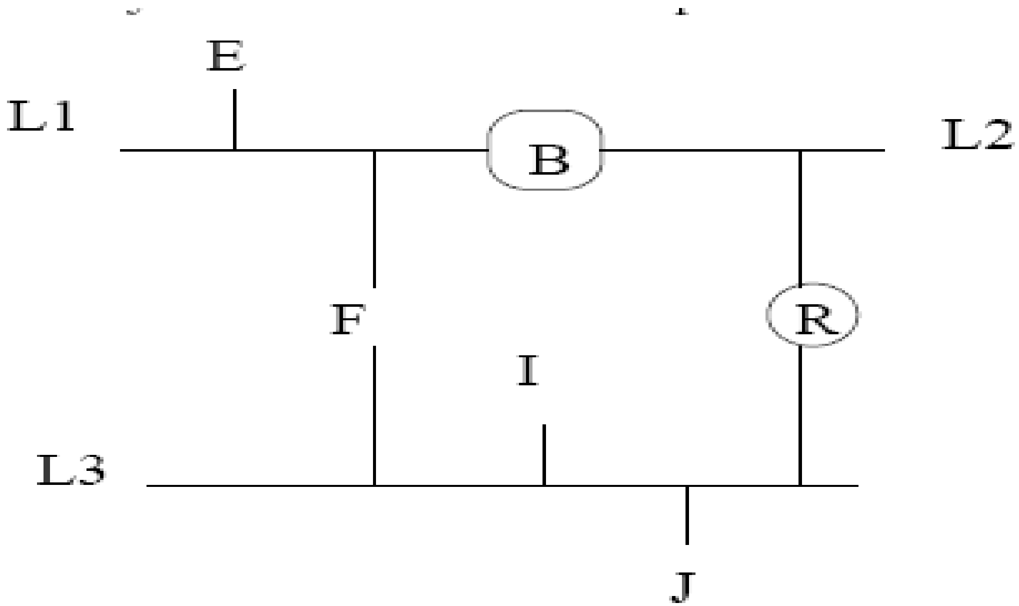
15-441 Computer Networking

Lecture 16 – Reliable Transport

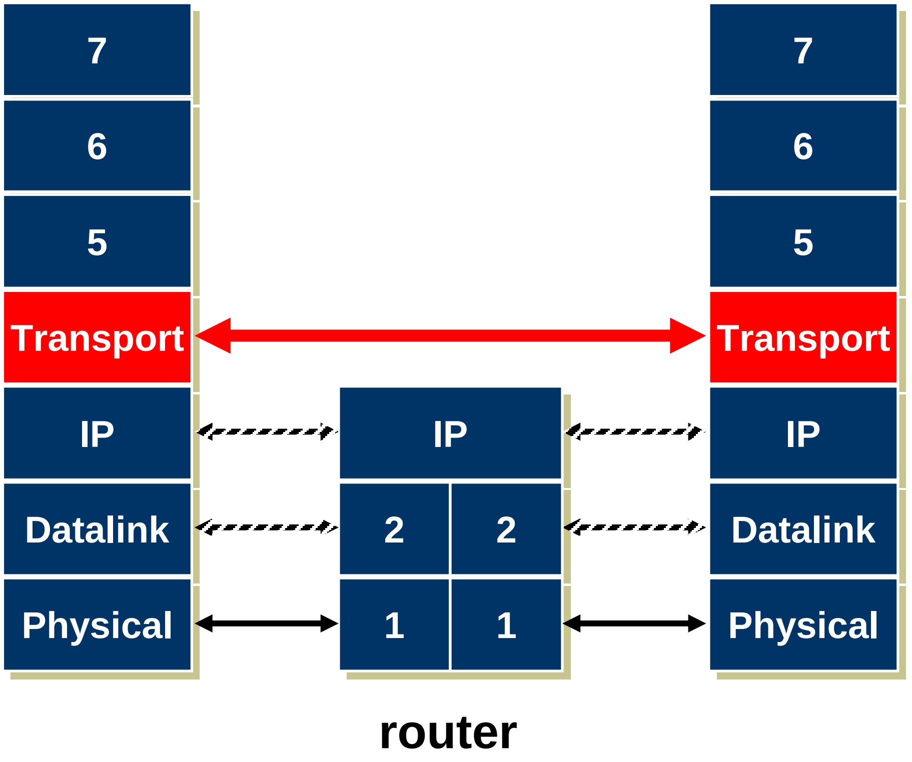
Last Question of Mid Term



Location	MAC Source	MAC Dest	IP Source	IP Dest
E to F on link L1	e	f[1]	E	F[1]
E to J on L1	e	r(2)	E	J
E to J on L2	e	r(2)	E	J
E to J on L3	r(3)	j	E	J

Transport Protocols

- Lowest level end-to-end protocol.
  - Header generated by sender is interpreted only by the destination
  - Routers view transport header as part of the payload



Functionality Split

- Network provides best-effort delivery
- End-systems implement many functions
  - **Reliability**
  - **In-order delivery**
  - Demultiplexing
  - Message boundaries
  - Connection abstraction
  - Congestion control
  - …

What Problems Reliable Transport Solution Try to Solve?

- Best effort network layer
  - Packets can get corrupted
  - Packets can get lost
  - Packets can get re-ordered

Mechanisms used in Reliable Transport

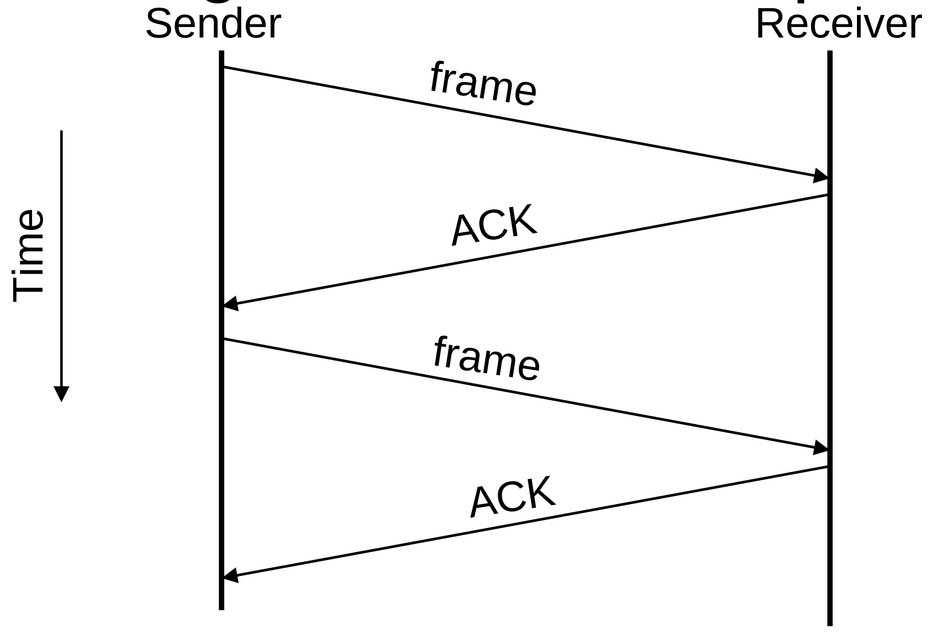
- Packets can get corrupted
  - CRC or Checksum to detect, retransmission to recover
  - Error correction code to recover
- Packets can get lost
  - Acknowledgement + Timeout to detect, retransmission to recover
- Packets can get re-ordered
  - Sequence number to detect, receiver buffer to re-order

Automatic Repeat Request (ARQ) Algorithms

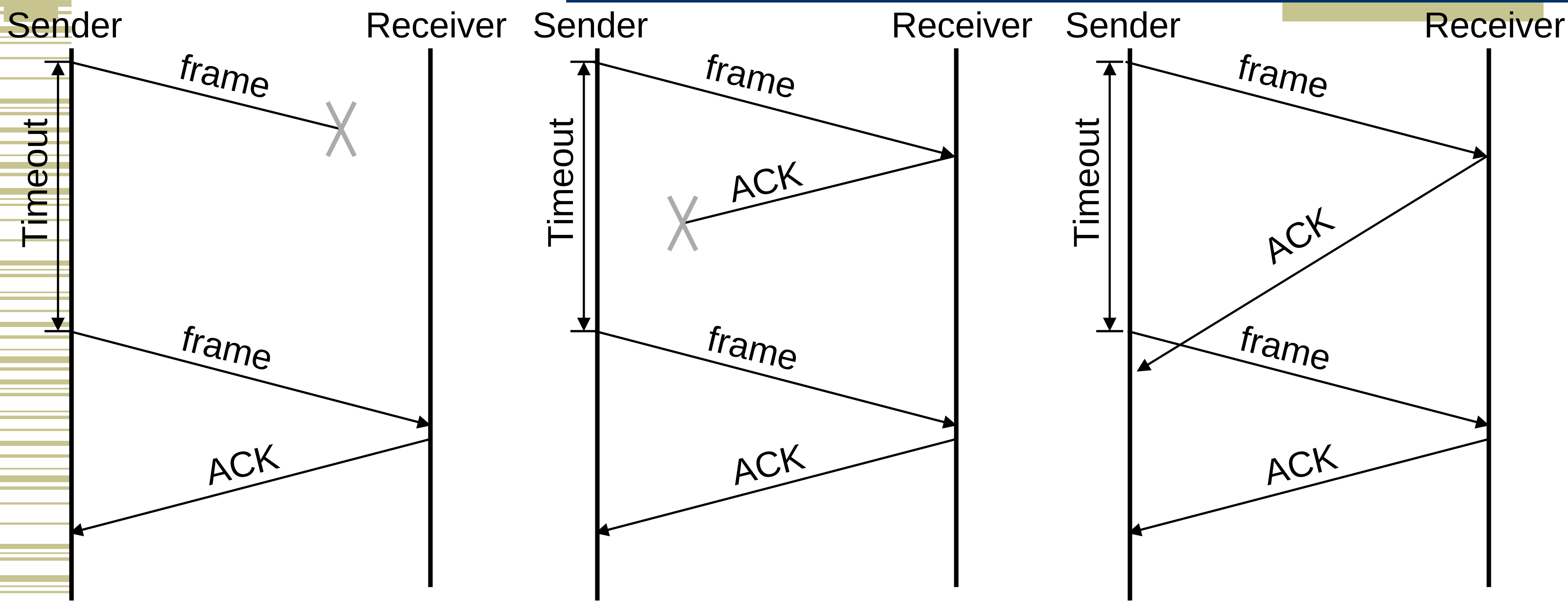
- Use two basic techniques:
  - Acknowledgements (ACKs)
  - Timeouts
- Two examples:
  - Stop-and-Wait
  - Sliding window

Stop-and-Wait

- Receiver: send an acknowledge (ACK) back to the sender upon receiving a packet (frame)
- Sender: excepting first packet, send a packet only upon receiving the ACK for the previous packet



What Can Go Wrong?



Frame lost → resent it on Timeout

ACK lost → resent packet

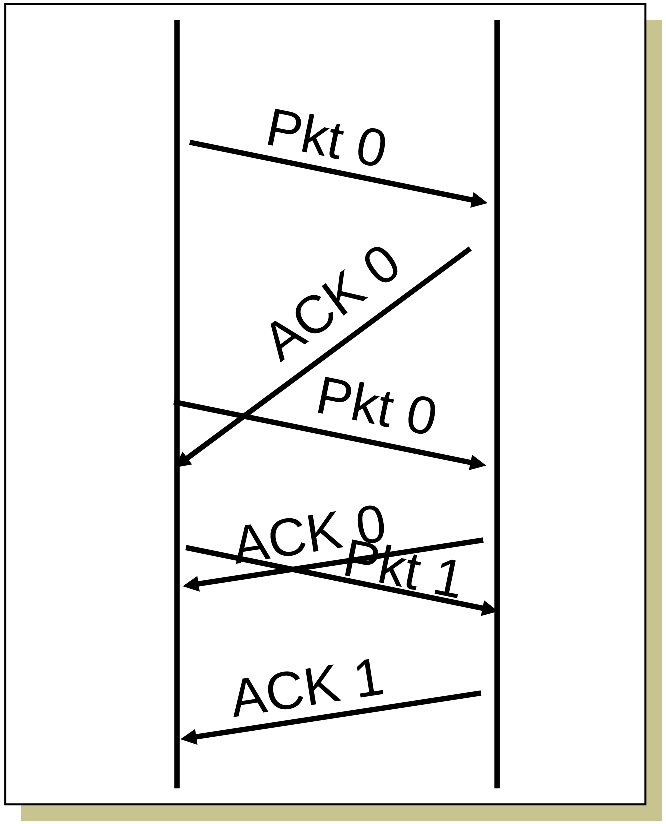
Need a mechanisms to detect duplicate packet

ACK delayed → resent packet

Need a mechanism to differentiate between ACK for current and previous packet

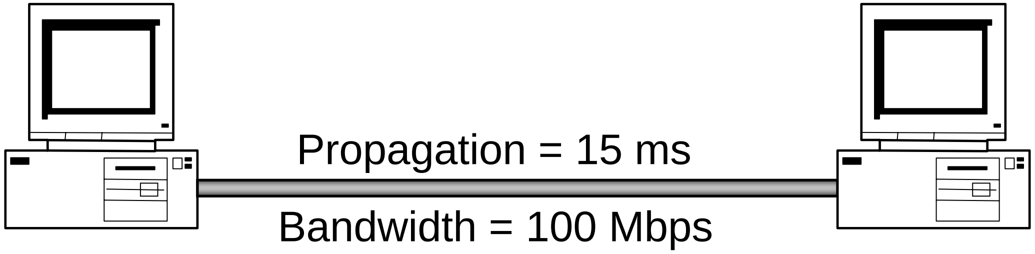
How to Recognize Retransmissions?

- Use sequence numbers
  - both packets and acks
- Sequence # in packet is finite → How big should it be?
  - For stop and wait?
- One bit – won't send seq #1 until received ACK for seq #0



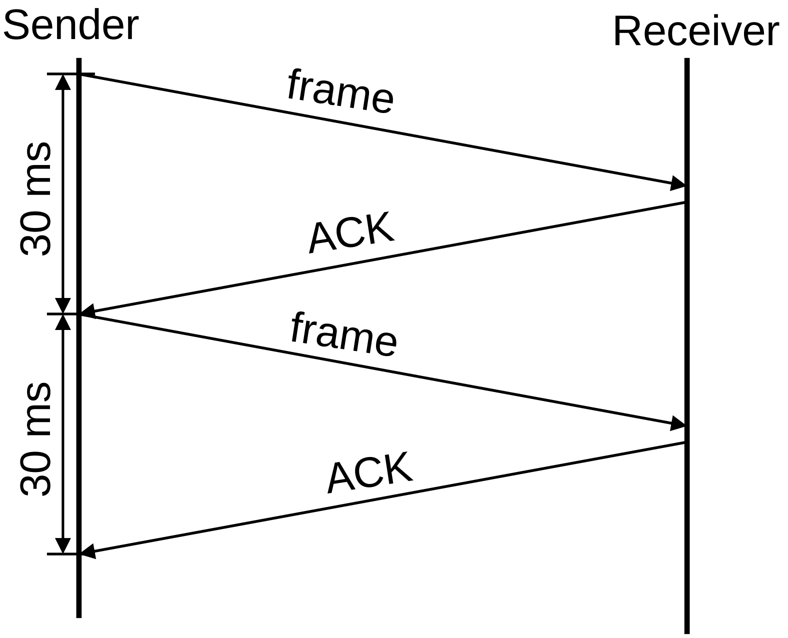
Stop-and-Wait Disadvantage

- May lead to inefficient link utilization
- Example: assume
  - One-way propagation = 15 ms
  - Bandwidth = 100 Mbps
  - Packet size = 1000 bytes → transmit = $(8*1000)/10^8$ = 0.08ms
  - Neglect queue delay → Latency = approx. 15 ms; RTT = 30 ms



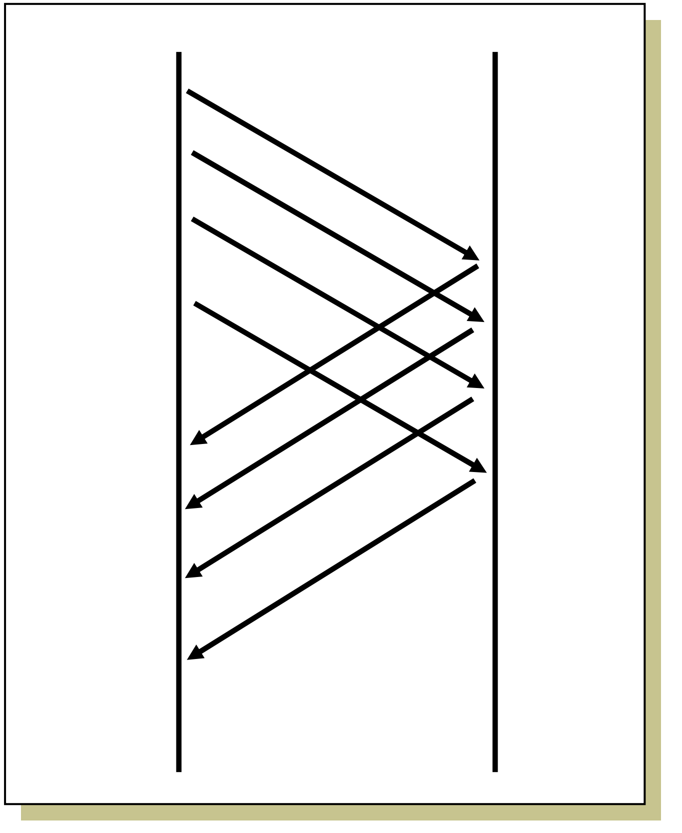
Stop-and-Go Disadvantage (cont'd)

- Send a message every 30 ms → Throughput = (8*1000)/0.03 = 0.2666 Mbps
- Thus, the protocol uses less than 0.3% of the link capacity!



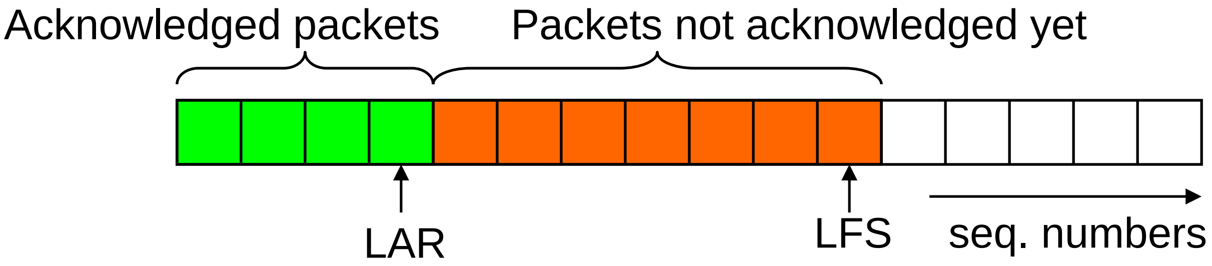
How to Keep the Pipe Full?

- Send multiple packets without waiting for first to be acked
  - Number of pkts in flight = window
- Reliable, unordered delivery
  - Several parallel stop & waits
  - Send new packet after each ack
  - Sender keeps list of unack'ed packets; resends after timeout
  - Receiver same as stop & wait
- How large a window is needed?



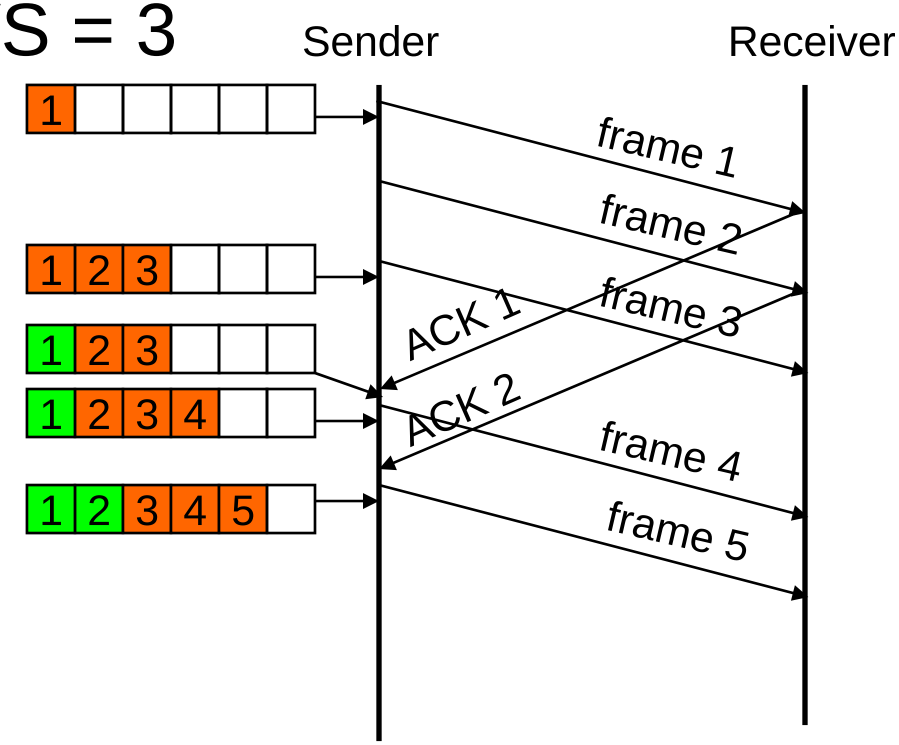
Sliding Window Protocol: Sender

- Each packet has a sequence number
  - Assume infinite sequence numbers for simplicity
- Sender maintains a window of sequence numbers
  - SWS (sender window size) – maximum number of packets that can be sent without receiving an ACK
  - LAR (last ACK received)
  - LFS (last frame sent)



Example

- Assume SWS = 3



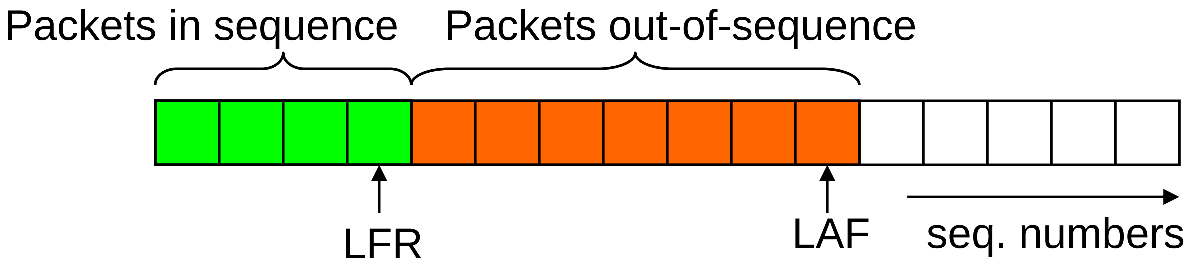
Note: usually ACK contains the sequence number of the first packet in sequence expected by receiver

Sliding Window Protocol: Receiver

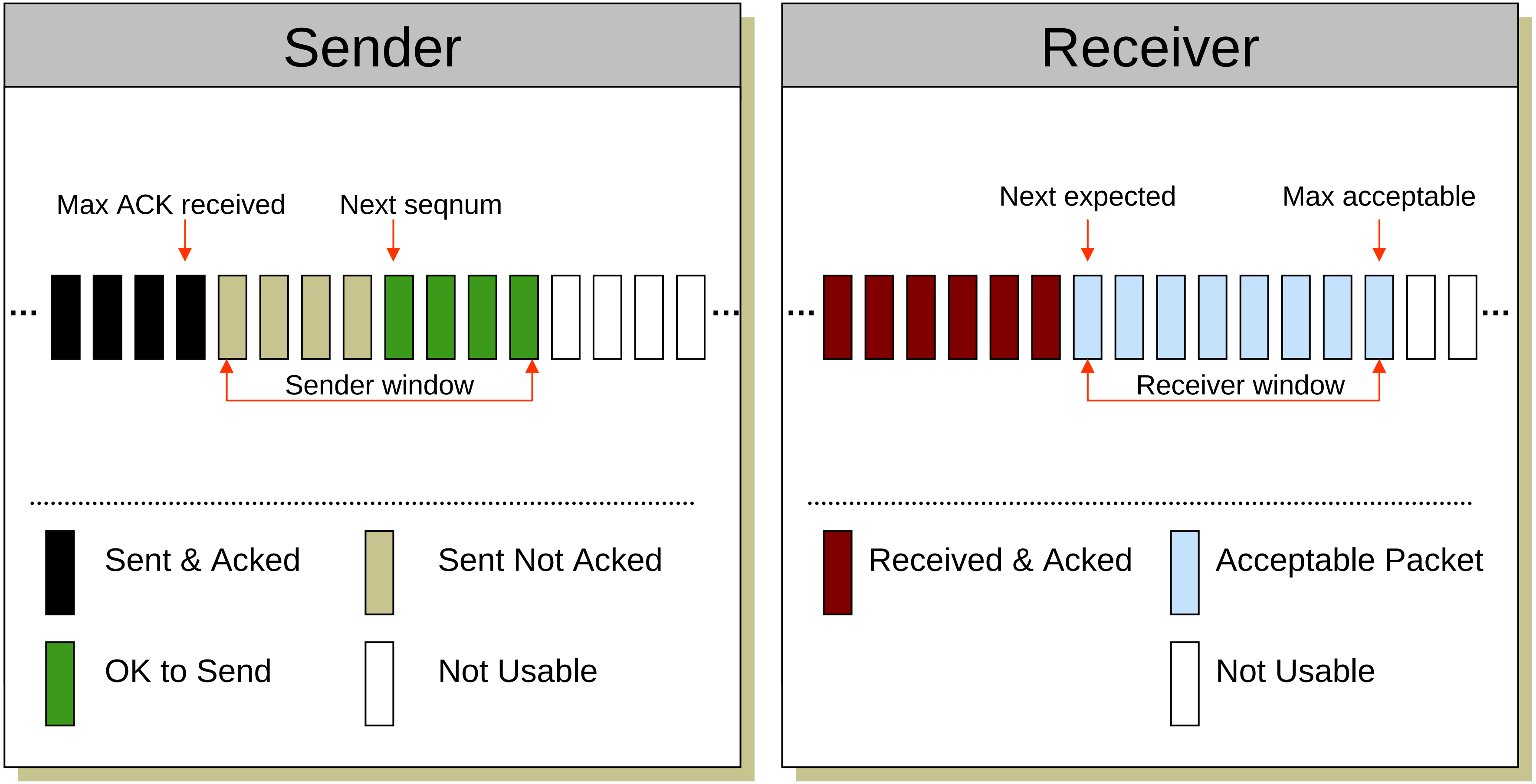
- Receiver maintains a window of sequence numbers
  - RWS (receiver window size) – maximum number of out-of-sequence packets that can received
  - LFR (last frame received) – last frame received in sequence
  - LAF (last acceptable frame)
  - LAF – LFR <= RWS

Sliding Window Protocol: Receiver

- Let seqNum be the sequence number of arriving packet
- If (seqNum <= LFR) or (seqNum >= LAF)
  - Discard packet
- Else
  - Accept packet
  - ACK largest sequence number seqNumToAck, such that all packets with sequence numbers <= seqNumToAck were received



Sender/Receiver State



Sequence Numbers

- How large do sequence numbers need to be?
  - Must be able to detect wrap-around
  - Depends on sender/receiver window size
- E.g.
  - Max seq = 7, send win=recv win=7
  - If pkts 0..6 are sent succesfully and all acks lost
    - Receiver expects 7,0..5, sender retransmits old 0..6!!!
- Max sequence must be $\geq$ send window + recv window

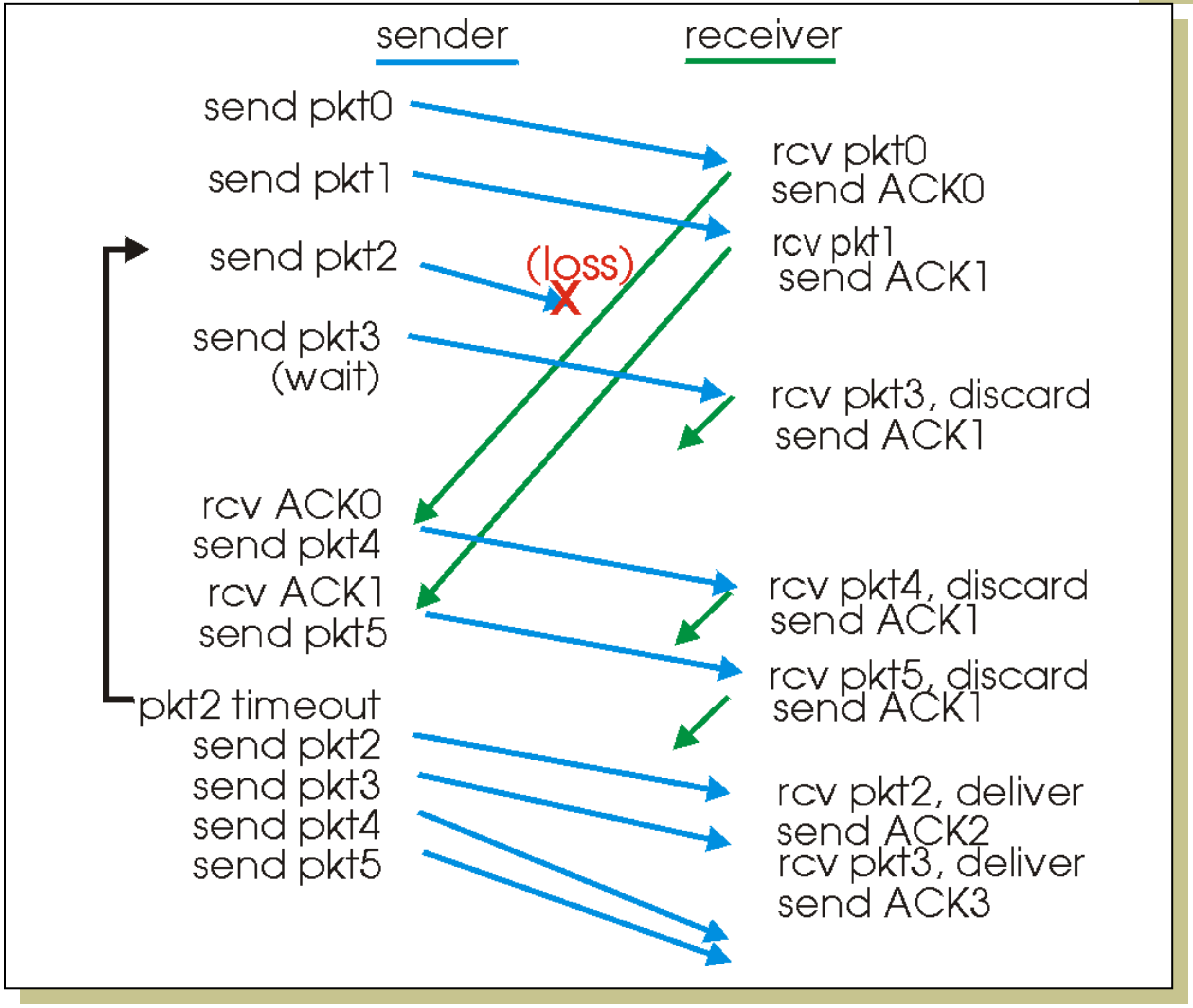
Cumulative ACK + Go-Back-N

- On reception of new ACK (i.e. ACK for something that was not acked earlier)
  - Increase sequence of max ACK received
  - Send next packet
- On reception of new in-order data packet (next expected)
  - Hand packet to application
  - Send cumulative ACK – acknowledges reception of all packets up to sequence number
  - Increase sequence of max acceptable packet

Loss Recovery

- On reception of out-of-order packet
  - Send nothing (wait for source to timeout)
  - Cumulative ACK (helps source identify loss)
- Timeout (Go-Back-N recovery)
  - Set timer upon transmission of packet
  - Retransmit all unacknowledged packets
- Performance during loss recovery
  - No longer have an entire window in transit
  - Can have much more clever loss recovery

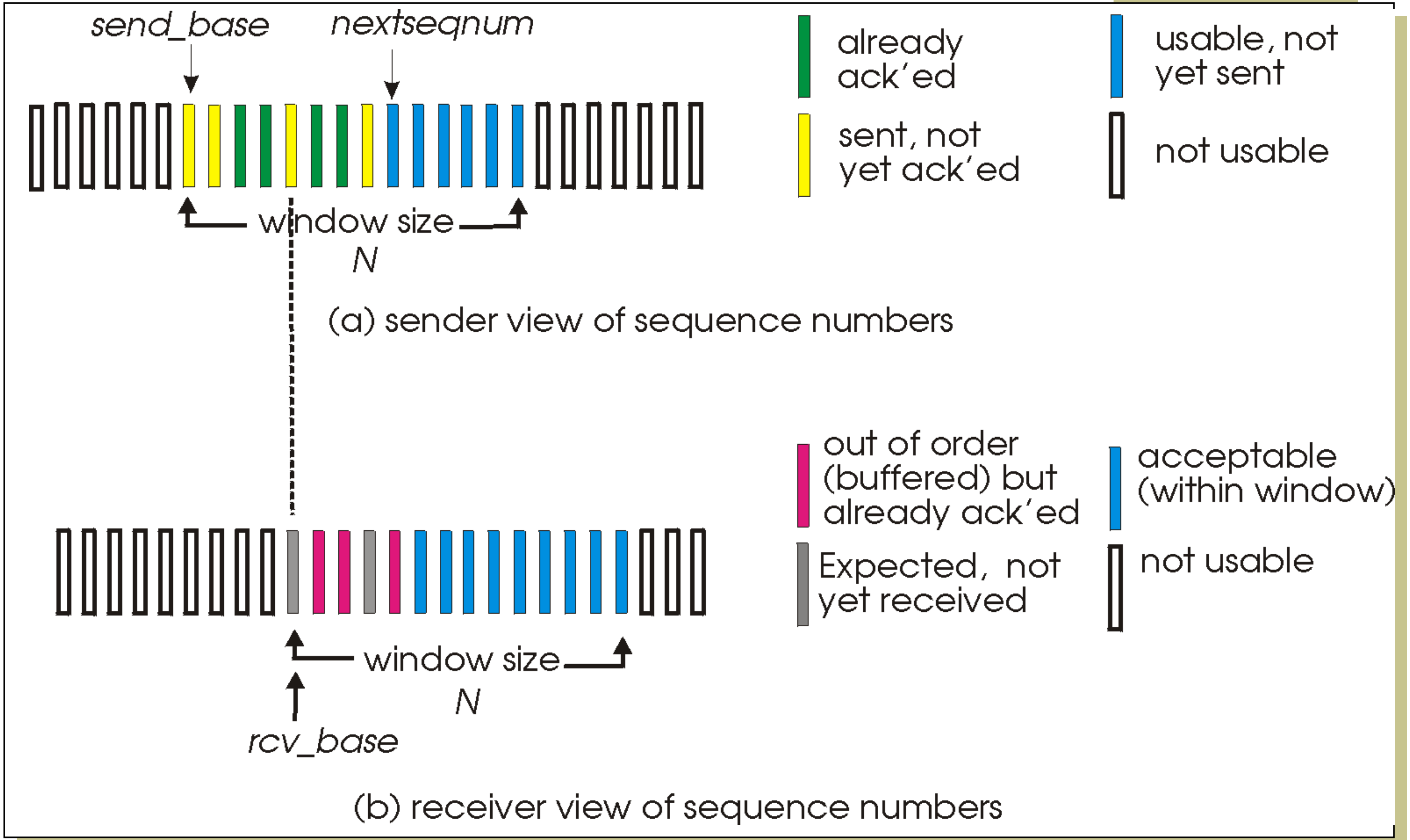
Go-Back-N in Action



Selective Ack + Selective Repeat

- Receiver *individually* acknowledges all correctly received pkts
  - Buffers packets, as needed, for eventual in-order delivery to upper layer
- Sender only resends packets for which ACK not received
  - Sender timer for each unACKed packet
- Sender window
  - N consecutive seq #'s
  - Again limits seq #s of sent, unACKed packets

Selective Repeat: Sender, Receiver Windows



Summary of ARQ Protocols

- Mechanisms:
  - Sequence number
  - Timeout
  - Acknowledgement
- Sender window: fill the pipe
- Receiver window: handle out-of-order delivery

Many Nuances

- What type of acknowledgements?
  - Selective acknowledgement
  - Cumulative acknowledgement
  - Negative acknowledgement
- How big should be the timeout value, SWS, RWS, sequence number field?
- Reliability mechanism used to implement other functions: flow control, congestion control
  - Function overloading introduces ambiguity and complexity