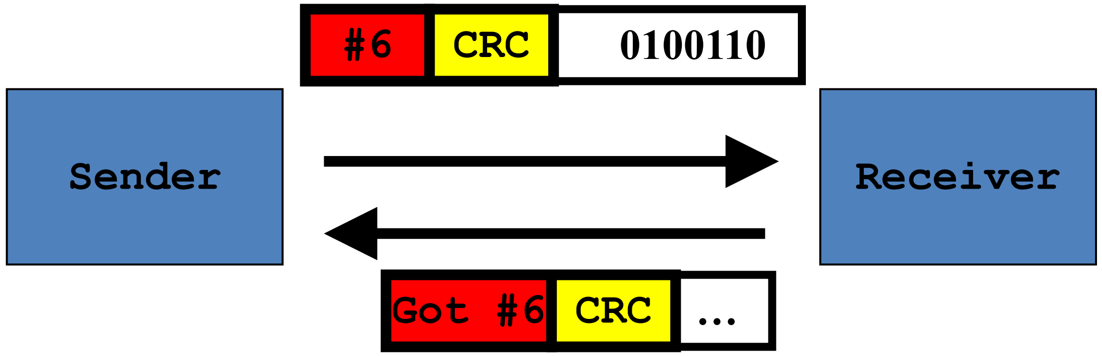


Reliable Protocol Design

Note: The slides are adapted from the materials from Prof. Richard Han at CU Boulder and Profs. Jennifer Rexford and Mike Freedman at Princeton University, and the networking book (Computer Networking: A Top Down Approach) from Kurose and Ross.

Review

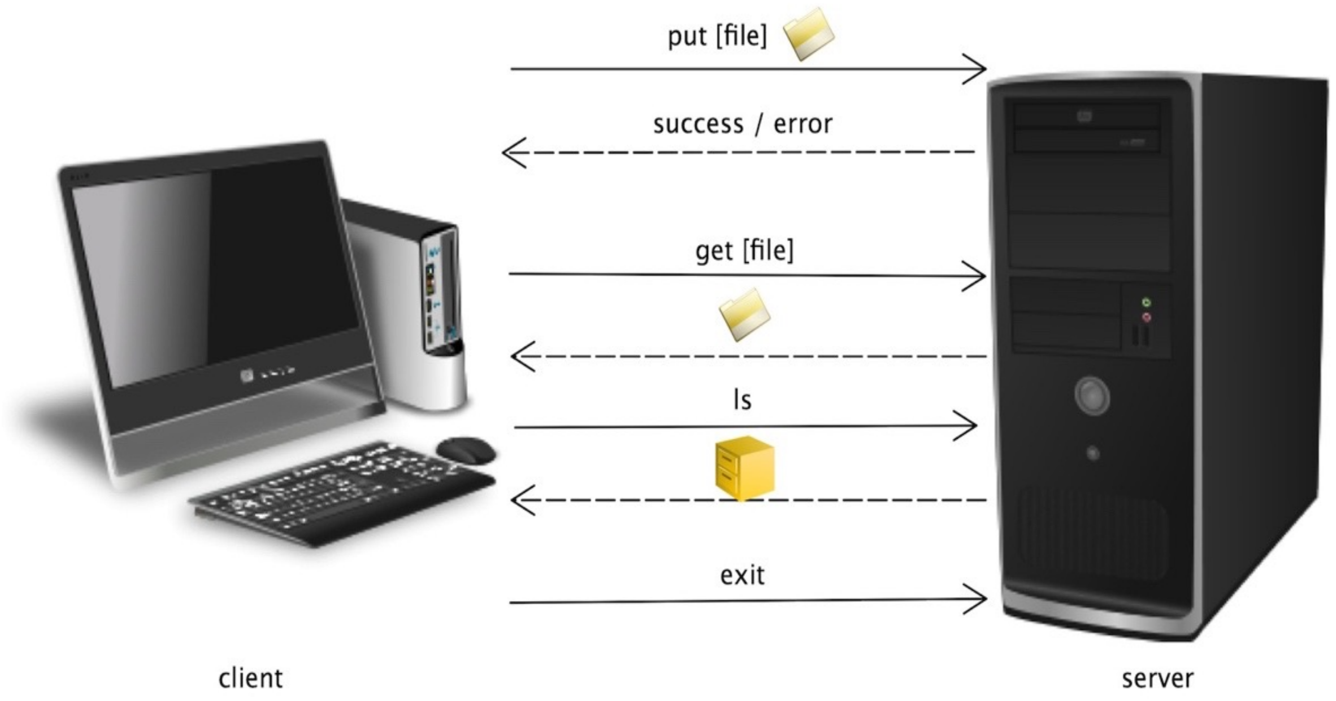
- Best Effort Packet Delivery
- End-to-End Principle
- Internet Stack (TCP/IP Model)
  - Application
  - Transport
  - Network
  - Link/Physical
- BDP (Bandwidth Delay Product)
- Sharing the Medium (Medium Access Control)

Announcements

- PA #1 is out, due 11:55 pm Feb 13 (Sun)

Programming Assignment #1

- UDP Socket Programming



Reliable Protocols

Recover from Losses

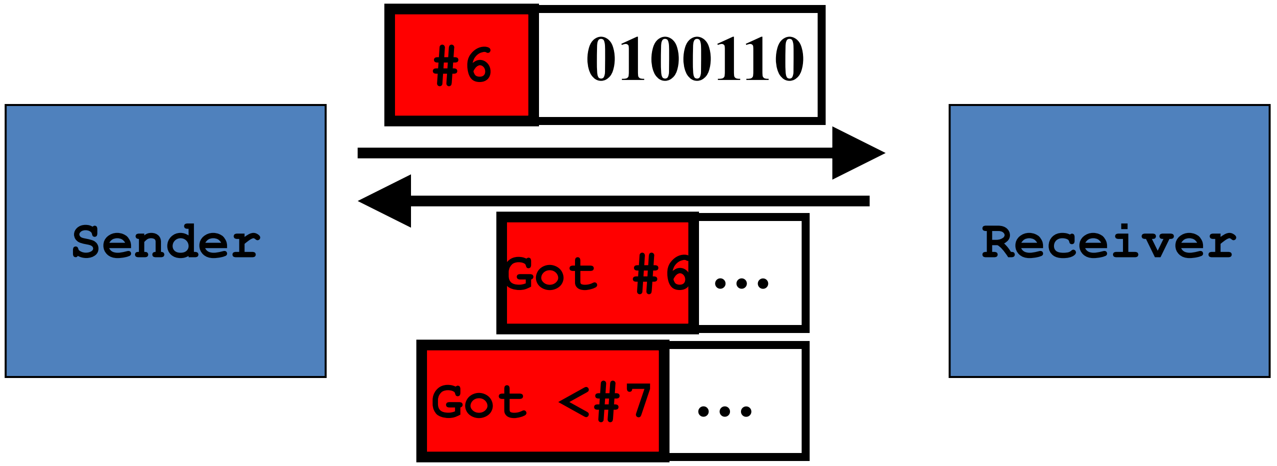
- Why aren't error detection and error correction enough?
  - Receiver drops a packet when errors detected
  - Receiver can't correct errors in some packets
  - Receiver never receives a packet
- How does the sender recover from such losses
  - Solution: Retransmit lost or corrupt packets
  - Also called ARQ (Acknowledgement-Repeat-Request) Protocols
  - Useful for communicating non-delay-sensitive data, e.g., Web pages, files, email, even playback video
  - Can incur too much delay for interactive audio/video

Acknowledgement

- How does the sender know a packet has been received properly
  - Analogy: Send a package by mail. How do I know it got there? Certified mail sends back a receipt
  - In reliable protocols, Acknowledgements are sent by receiver back to sender confirming receipt of a packet
- How does the sender know which packet is being acknowledged?
  - Sender labels each packet with a sequence number and increments it when an acknowledgement arrives at the sender. When receiver sees packet #6, it sends an acknowledgement with ID #6
  - Sequence #'s are also useful to send many packets at once

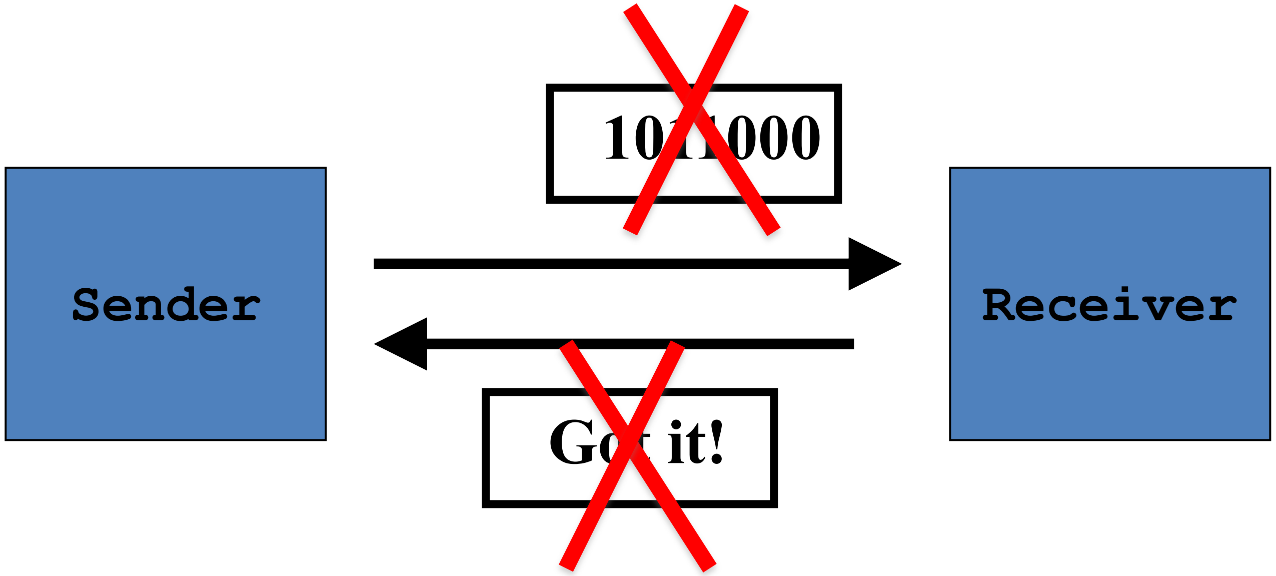
ACK's and NAK's

- Types of Acknowledgements
  - ACK's: positive acknowledgements ("I've received these packets")
    - Cumulative
    - Selective
  - NAK's: negative acknowledgements ("I have not received these packets")
  - ACK's are more prevalent than NAK's, why?



ACK can be lost too..

- If the packet or ACK is lost, how does the protocol sender decide when to retransmit?
  - It sets a timer as soon as the packet is transmitted. If the ACK hasn't arrived after a timeout, then the packet is retransmitted

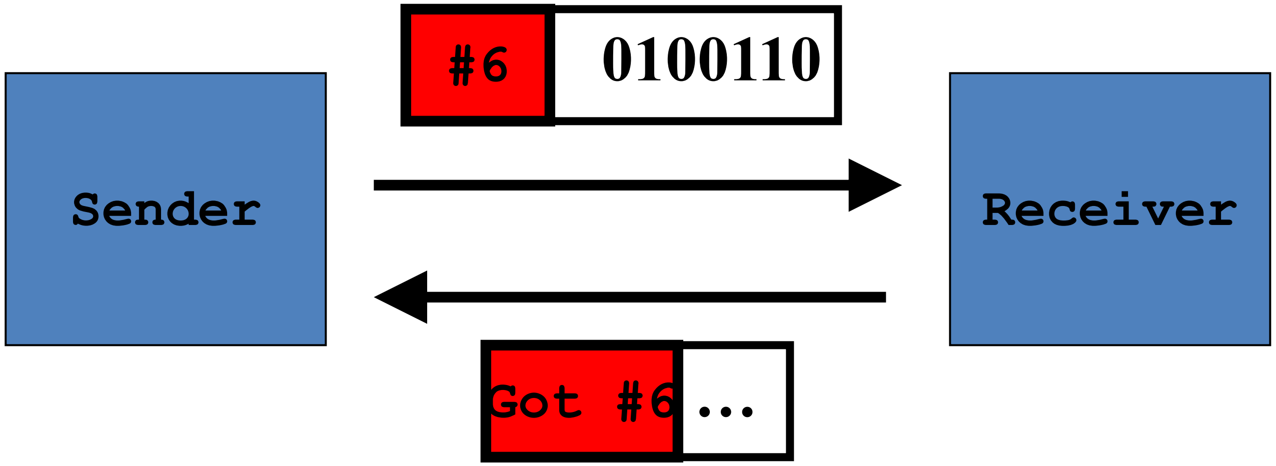


Timeouts

- How would you choose a good value for a timeout?
  - If timeout is too long, then waste bandwidth and send slowly
  - If timeout is too short, then retransmit unnecessarily
  - So choose timeout approximately equal to round-trip time RTT

Reliable Protocols Thus Far

- To detect when a packet needs to be retransmitted, reliable/ARQ protocols must use both:
  - Acknowledgements, and
  - Timeouts
- Sequence numbers:
  - Sender labels packets with them
  - Receiver labels packets with them

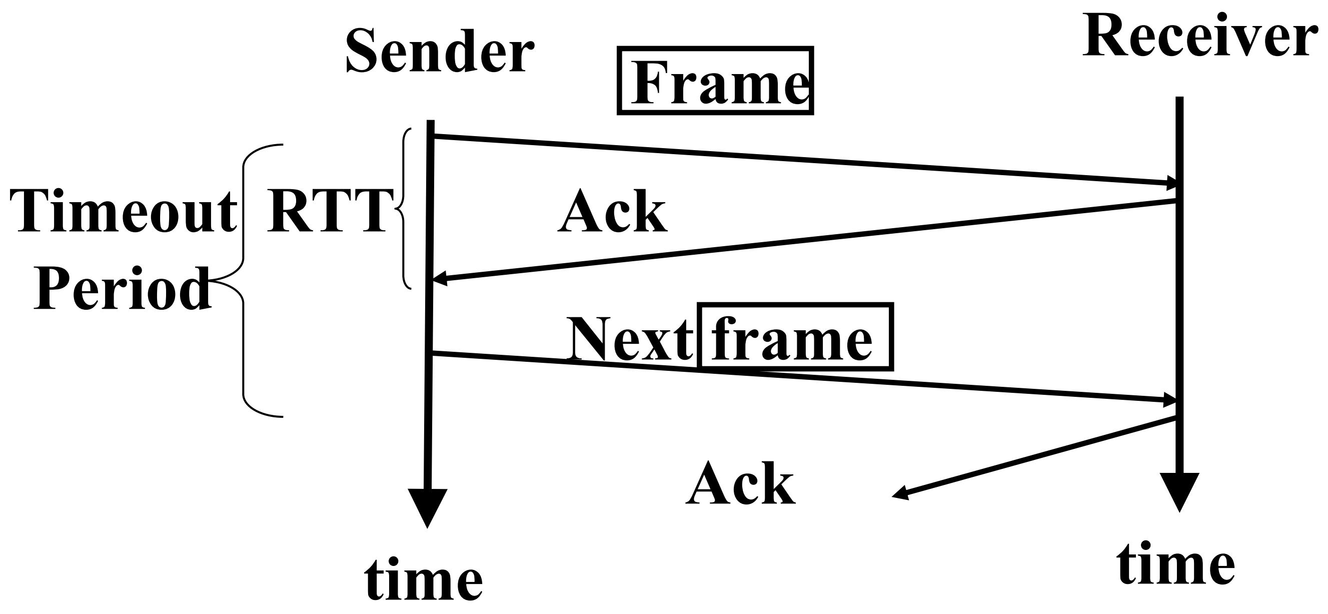


Improve Efficiency "Keep the Pipe Full"

- Stop-and-Wait
  - After transmitting a packet/frame, the sender stops and waits for an ACK before transmitting the next frame
  - If a timeout occurs before receiving an ACK, the sender retransmits the frame
- Go-Back-N
  - Maintain a sliding window at both sender and receiver of unacknowledged packets
- Selective Repeat (SRP)
  - Selective ACK's sent by receiver identify specifically which frame(s) have been received correctly

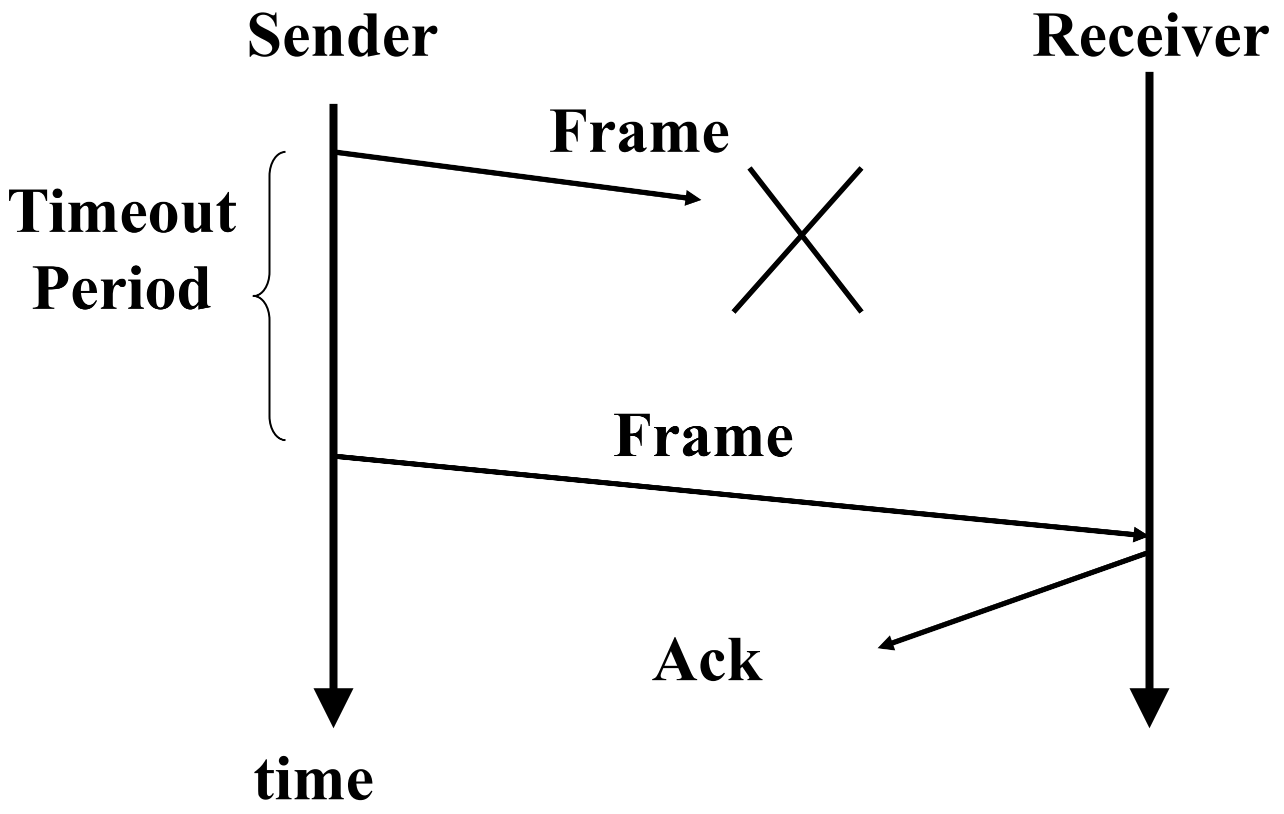
Stop-and-Wait Protocol

- Stop-and-Wait
  - After transmitting a packet/frame, the sender stops and waits for an ACK before transmitting the next frame
  - If a timeout occurs before receiving an ACK, the sender retransmits the frame



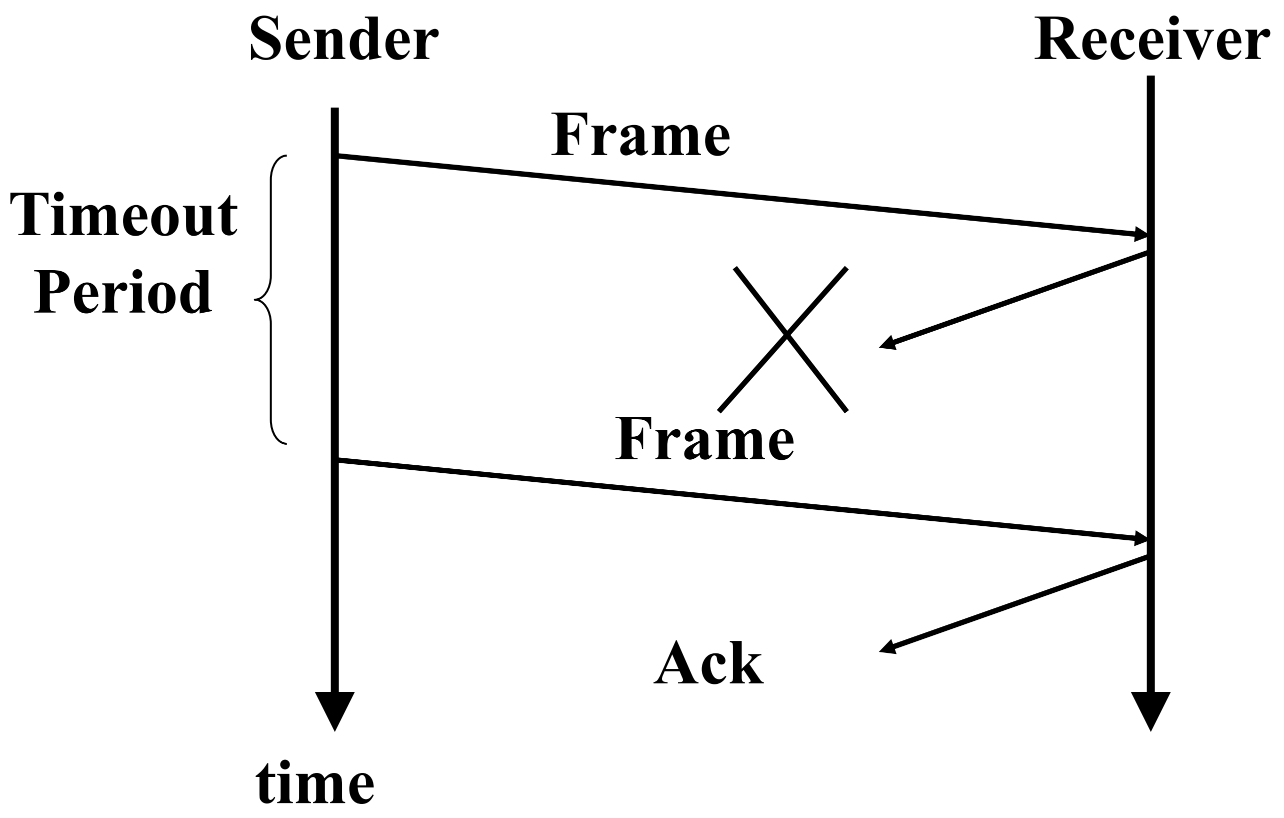
Stop-and-Wait Protocol (2)

- Stop-and-Wait
  - If a timeout occurs before receiving an ACK, the sender retransmits the frame



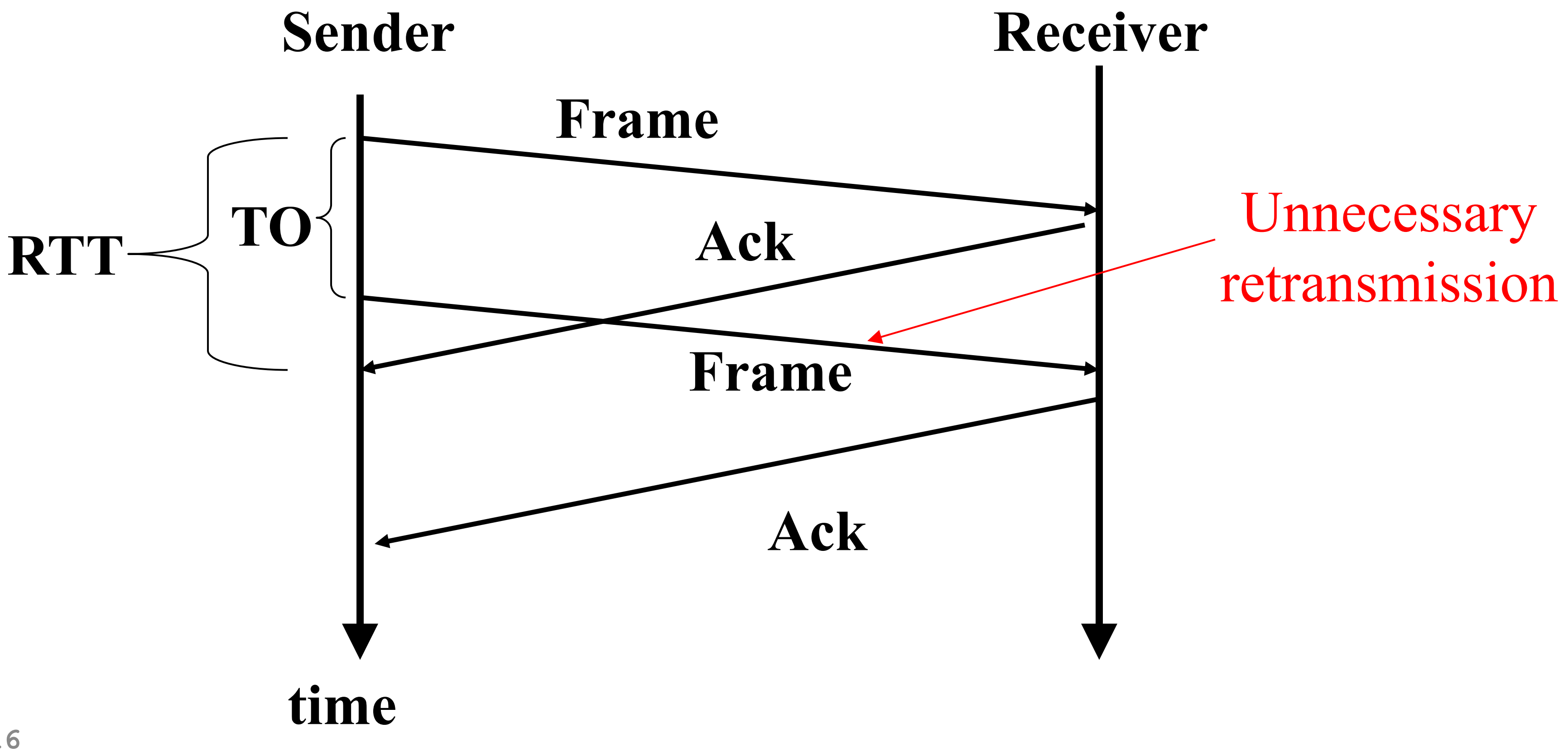
Stop-and-Wait Protocol (3)

- Stop-and-Wait
  - If a timeout occurs before receiving an ACK, the sender retransmits the frame



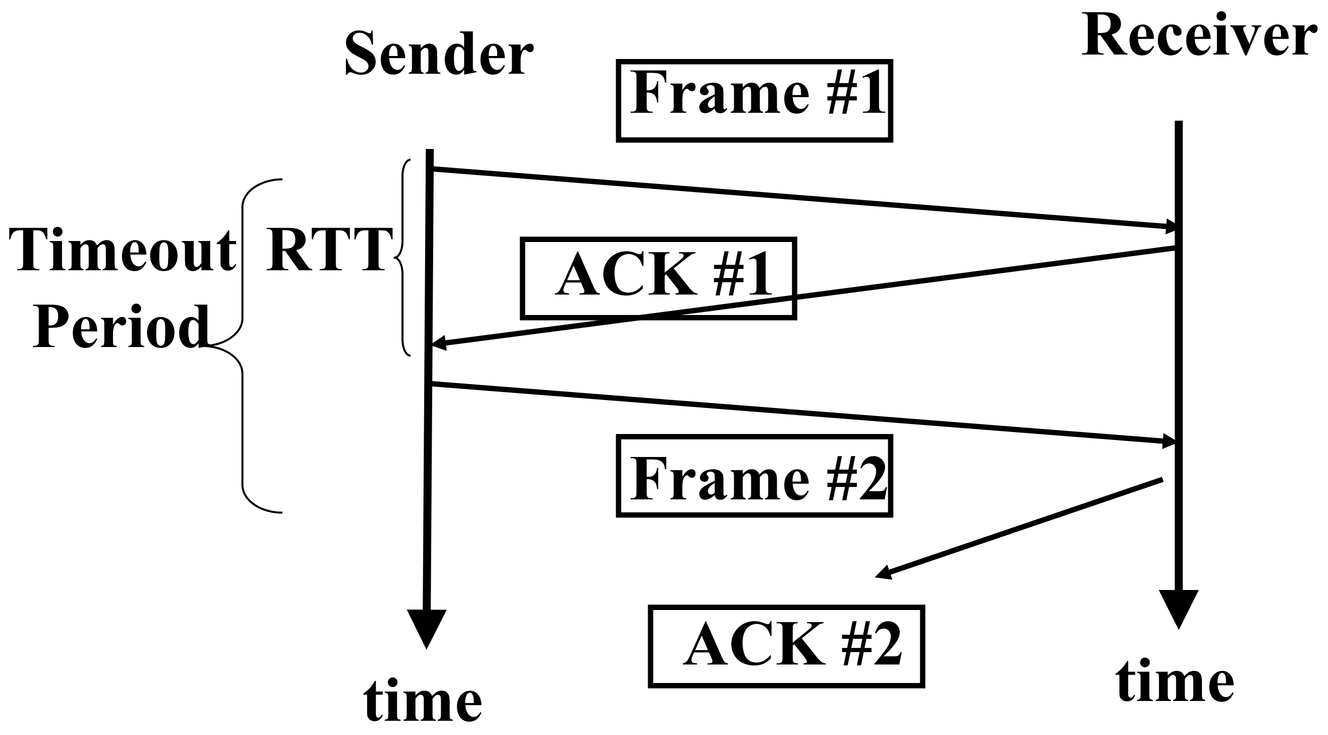
Stop-and-Wait Protocol (4)

- Stop-and-Wait
  - Want timeout >= RTT to avoid spurious retransmissions of a frame/packet



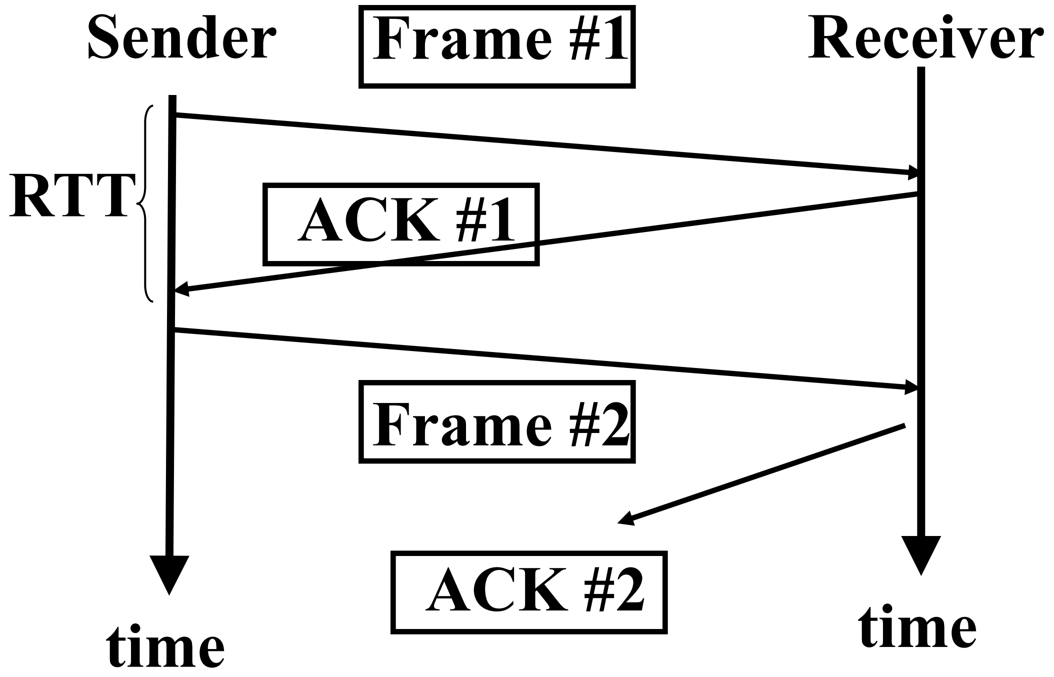
Stop-and-Wait Protocol (5)

- Stop-and-Wait: Why we need sequence #?
  - Label each packet and ACK with the proper sequence # to avoid confusion at receiver and sender (e.g., when ACK #1 is lost, Receiver does not know the retransmitted Frame #2 is for Frame #1 or Frame #2)



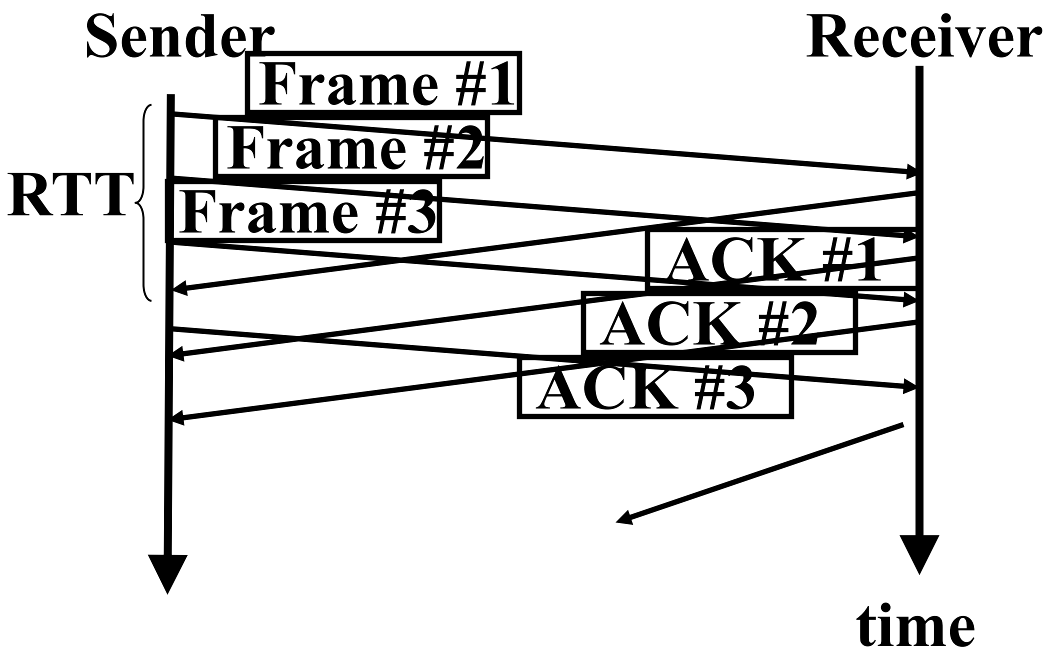
Problem with Stop-and-Wait

- Only one outstanding packet at a time = waste of link bandwidth
  - Example: 1.5Mbps link with RTT 45ms (BDP is 67.5Kb ≈ 8 KB pipe size)
  - If frame size is 1KB, then use only 1/8 of bandwidth



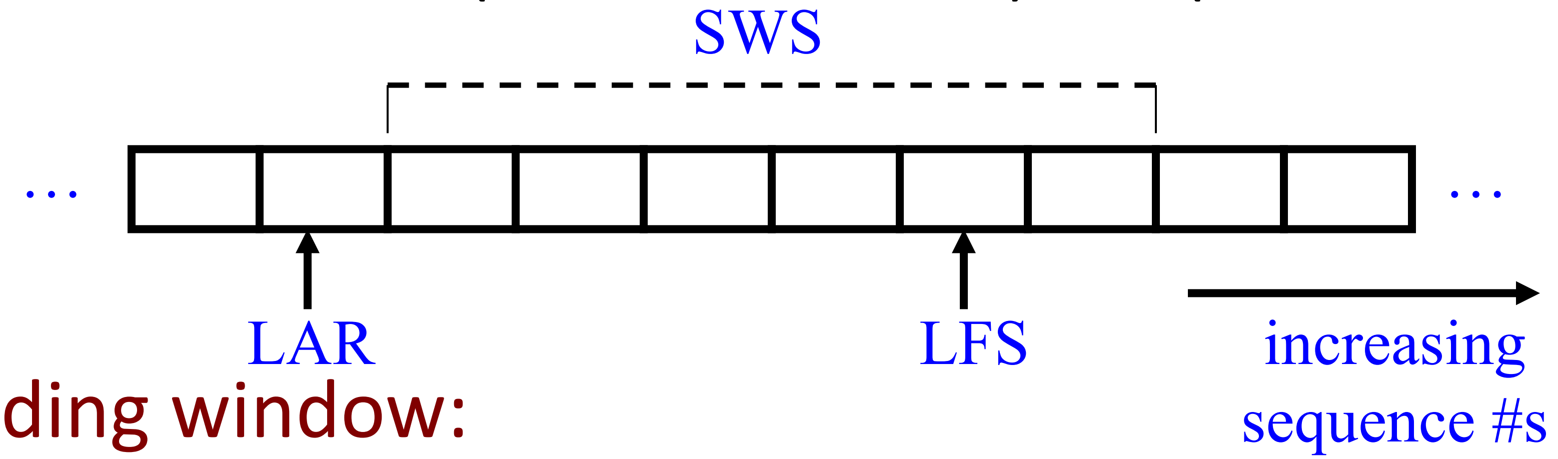
“Keep the Pipe Full”

- During one RTT, send N packets
  - Example: 1.5Mbps link with RTT 45ms (BDP is 67.5Kb ≈ 8 KB pipe size)
  - If frame size is 1KB, and N=3, then can have 3 outstanding packets, 3/8 of BW, and triple bandwidth utilization



Go-Back-N Sliding Window Protocol

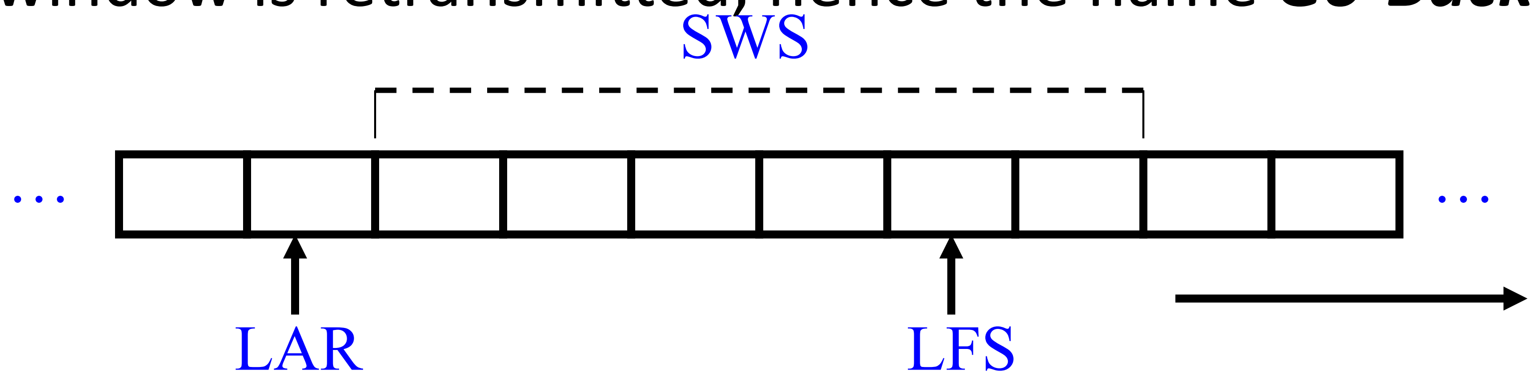
- Maintain a sliding window at both sender and receiver of unacknowledged packets
  - Send Window Size (SWS)
  - At sender: LAR (last ACK received), LFS (last frame sent)



- Sliding window:
  - When ACK # (LAR + 1) arrives, slide LAR to right
  - LFS – LAR <= SWS

Go-Back-N Sliding Window Protocol (2)

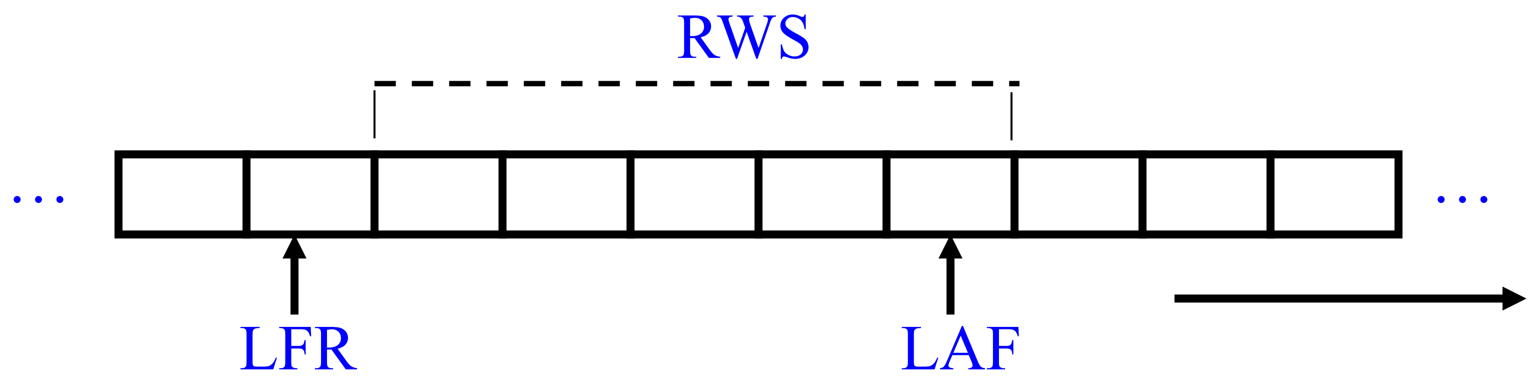
- Timeouts and retransmissions
  - Each frame in the window has its own timeout
  - When the timeout for frame #(LAR+1) expires, the entire window is retransmitted, hence the name ***Go-Back-N***



  - Each frame needs its own timeout, because if many ACKs arrive, then LAR slides forward to the last unACKed packet and we need to know how long that packet has been in the pipe
  - Frame # (LAR+1)'s timeout always expires first – so book's statement that a frame is retransmitted when its time expires is accurate

Go-Back-N Sliding Window Protocol (3)

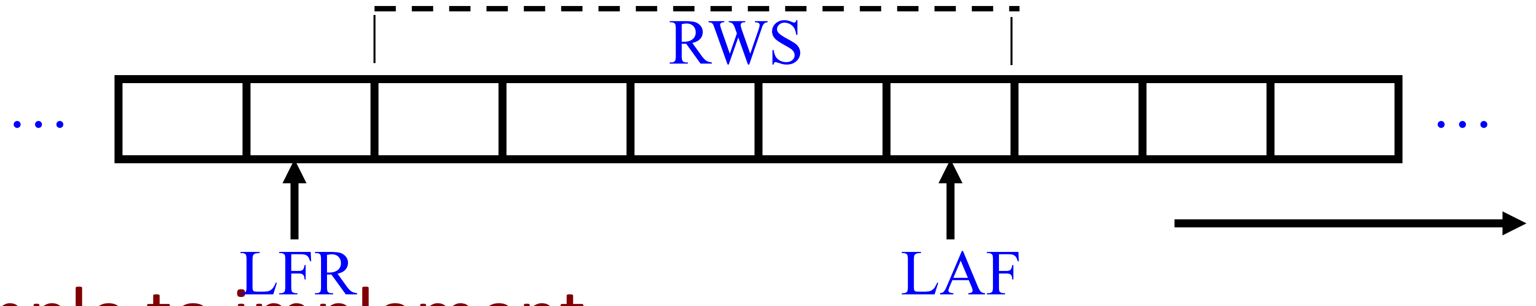
- At receiver:
  - Maintain a Receive Window Size (RWS)
  - LAF (largest acceptable frame), LFR (last frame received)



- Sliding window:
  - When frame arrives, keep it if its within window
  - If frame # (LFR+1) arrives, then slide window to right (increment LFR and LAF)
  - Send back Cumulative ACK = LFR, LAF-LFR <=RWS

Go-Back-N Sliding Window Protocol (4)

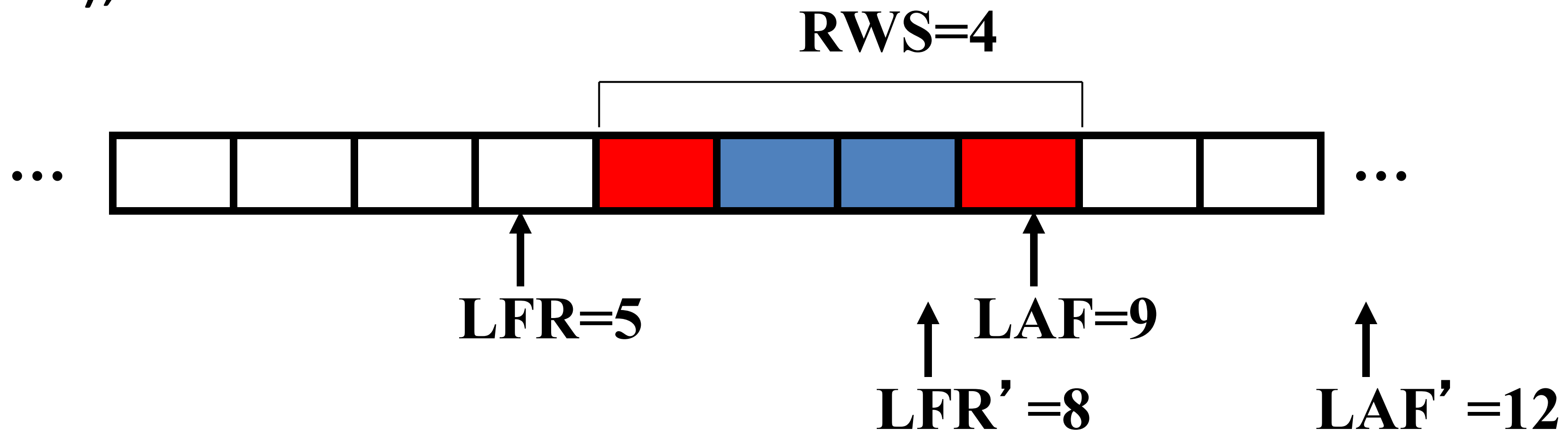
- A cumulative ACK = LFR says that everything up to the acknowledged sequence number has been received



- Simple to implement
- If you received an out-of-order packet, you can
  - Data: keep or drop it
  - ACK: cumulative ACK each time an (out-of-order) packet arrives, or wait until you can cumulatively ACK a new packet
- Cumulative ACK can also be interpreted at sender as the next packet the receiver expects, rather than the last packet the receiver got

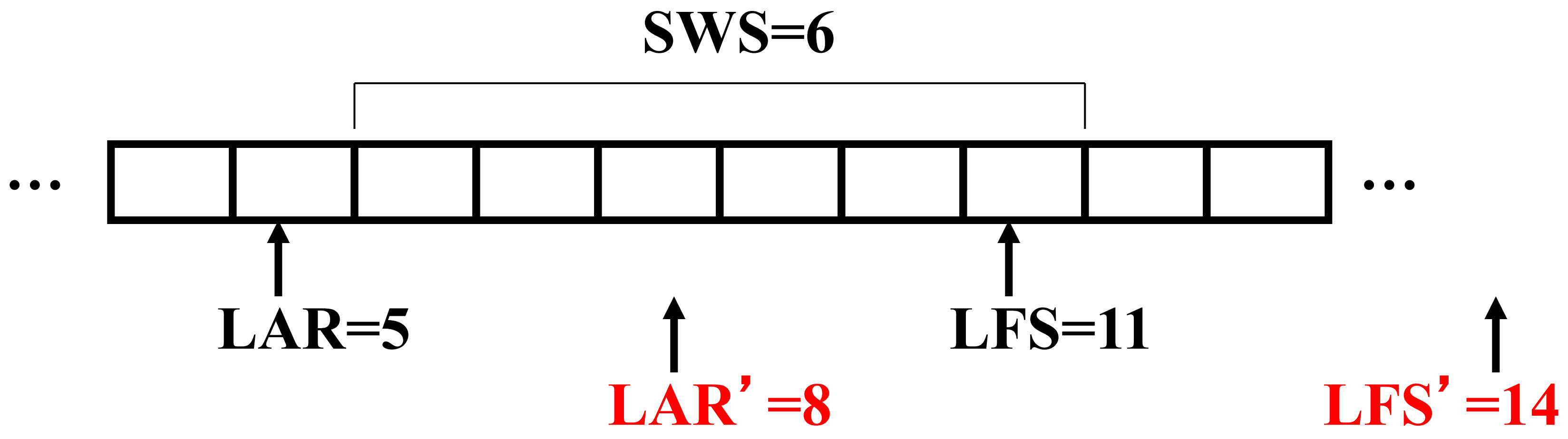
Go-Back-N Sliding Window Protocol (5)

- Example: RWS = 4, LFR = 5, LAF = 9
  - Suppose at receiver frames #7 and #8 arrive, but frames #6 or #9 either arrive out of order or are lost. Let's keep the data frames #7 and #8
  - When frame #7 arrives out of order, then send cumulative ACK with sequence #5 (same for frame #8)
  - When frame #6 arrives, slide window (LFR' = 8, LAF' = 12), and send cumulative ACK = #8



Go-Back-N Sliding Window Protocol (6)

- Meanwhile, back at the sender …
  - Example: suppose SWS=6, LAR = 5, LFS = 11
  - Each time sender gets a cumulative ACK of #5, it waits. If timeout, retransmits frame #6 through frame #11
  - When sender gets cumulative ACK #8, it slides window right (LAR' = 8, LFS' = 14)



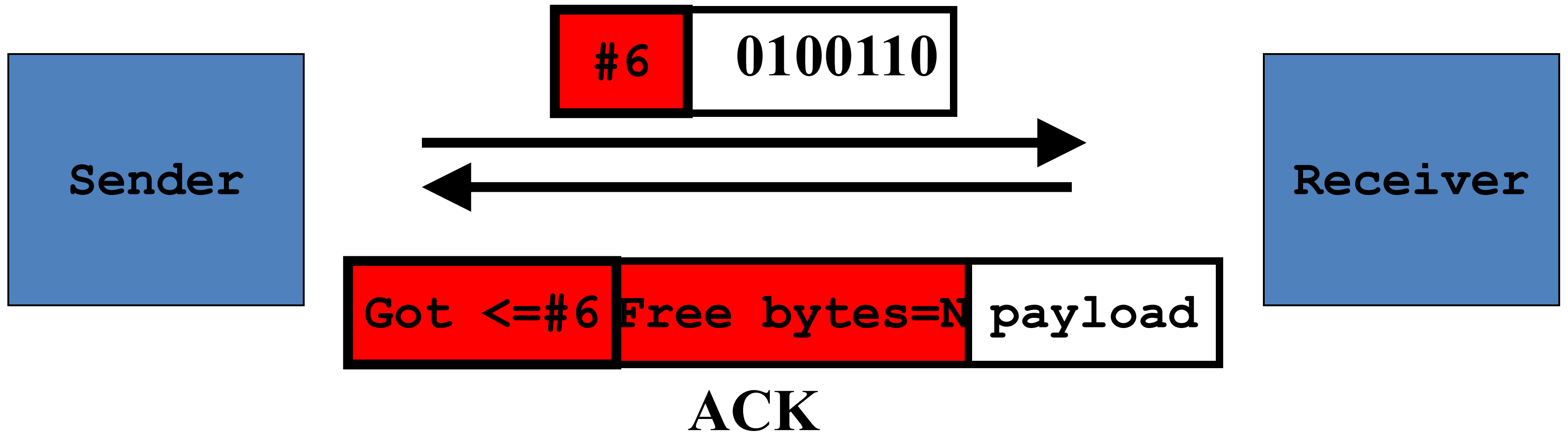
Go-Back-N Sliding Window Protocol (7)

- In previous example, SWS=6 <> RWS=4
  - – Sender's effective window size is 4!
  - – In general, effective window size is min(SWS, RWS)
- What is the optimal window size?
  - – BDP (full utilization of the pipe)
  - – But there's one more factor – receiver devotes variable CPU time to packet processing => flow control
  - – But wait, there's another factor – network's bandwidth can fluctuate too => congestion control
    - BW over shared media can fluctuate
    - Internet BW can fluctuate – TCP congestion control

Window-Based Flow Control

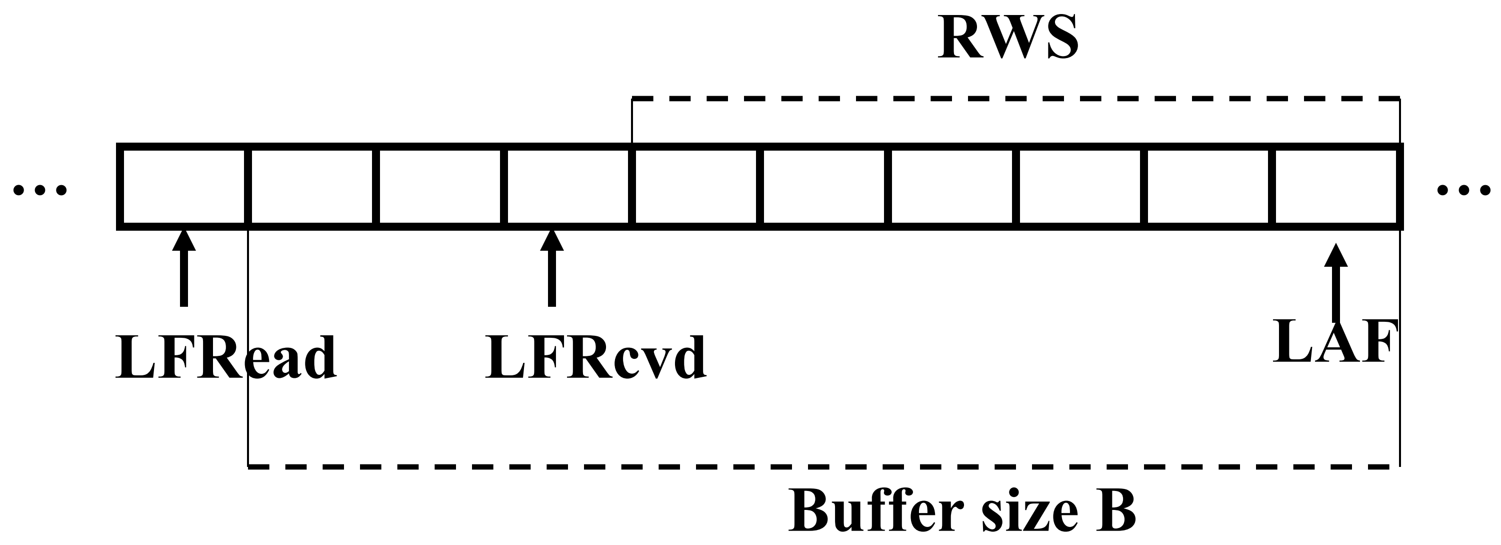
- Flow control deals with mismatch in rates between sender and receiver
  - If receiver is slow, then sender should slow down to match the receiver
- Two forms of flow control:
  - Rate-based flow control
    - If receiver can only process packets at a rate of X bits/sec, then construct a filter at sender that limits transmission to X bps
  - Window-based flow control
    - Receiver sends advertisements of its free receive buffer size back to sender
    - Sender limits itself to sending only as much as the receiver can currently handle

Window-Based Flow Control (2)



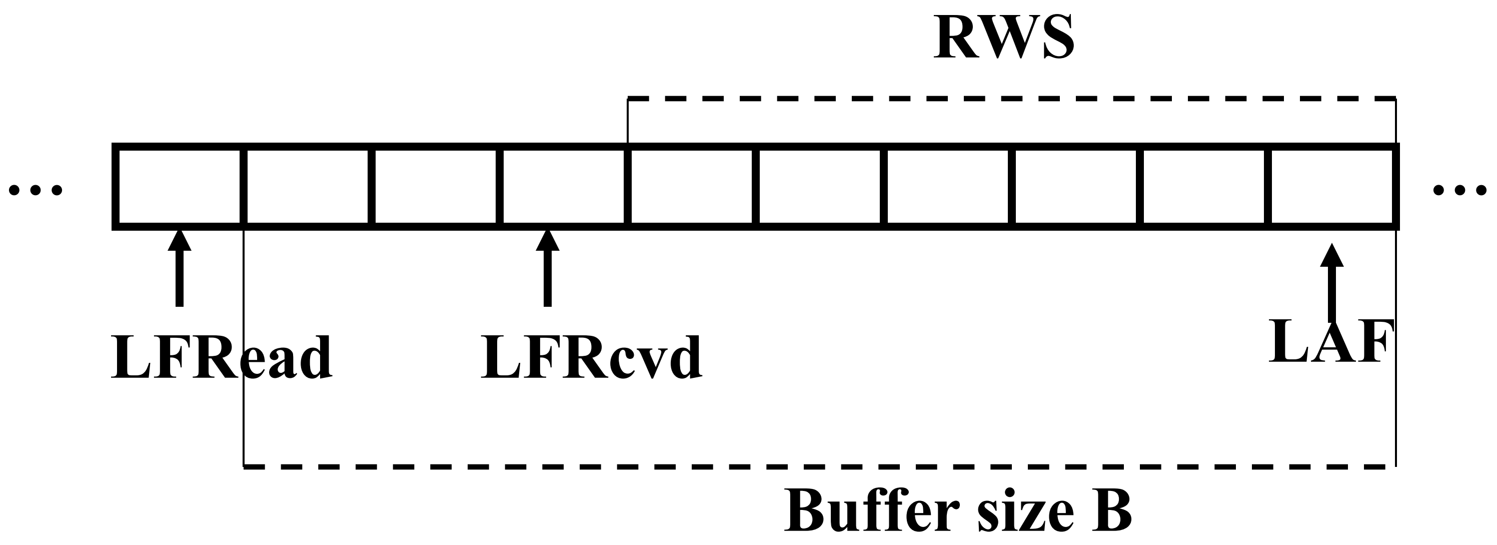
- Receiver allocates a finite buffer size B to receive data
  - RWS <= B
  - Application is slow to read data from buffer, so let LFRead = last frame read by application from buffer
  - To avoid confusion, rewrite LFR and LFRcvd

Window-Based Flow Control (3)



- Application has only read data up to LFRead
- In meantime, sliding window protocol has moved on and continues to reliably deliver more data
  - Upper limit = LFRead + B
  - LFRcvd <= upper limit, Largest Acceptable Frame LAF = upper limit
- Amount of Free Space = (upper limit) - LFRcvd

Window-Based Flow Control (4)



- Set RWS = Amount of Free Space = LFRead + B – LFRcvd
  - This also sets LAF = LFRcvd + RWS
  - Initially, set RWS = B
- Put RWS as the window advertisement in the ACK and transmit ACK to sender
- Sender sets its SWS = advertised RWS

Window-Based Flow Control (5)

- What happens if RWS drops to zero?
  - Sender stops sending, sets SWS = RWS = 0
- What problem could occur when RWS = 0?
  - Could wind up in a deadlocked state
    1. Normally the receiver only generates ACKs when data arrives – so receive waits on sender for data
    2. But , sender stops sending any packets when it receives a zero window advertisement – so sender is waiting on the receiver for a new ACK
    - DEADLOCK due to 1 and 2
    - Even if the receiver window opens up, the receiver's new ACK (and all such window-opening ACKs) may be lost – still get DEADLOCK

Window-Based Flow Control (6)

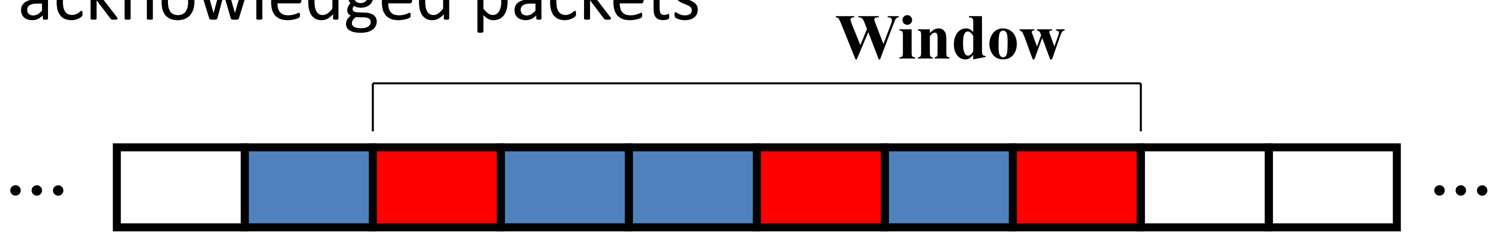
- Solutions to flow control deadlock:
  - When RWS = 0, Receiver periodically generates an ACK if the free space window opens (Application reads data & frees up space) until new data arrives (must be periodic because a single ACK could be lost)
  - When RWS = 0, sender generates periodic probe data packets (TCP uses this solution)

Problem with Go-Back-N

- Why are cumulative ACK’s considered inefficient?
  - Cumulative ACK’s causes unnecessary retransmissions => inefficient
  - In our example, packets #7 and #8 are retransmitted even though they’ve already arrived at receiver
- Solution: acknowledge each packet individually, or selectively, rather than cumulatively

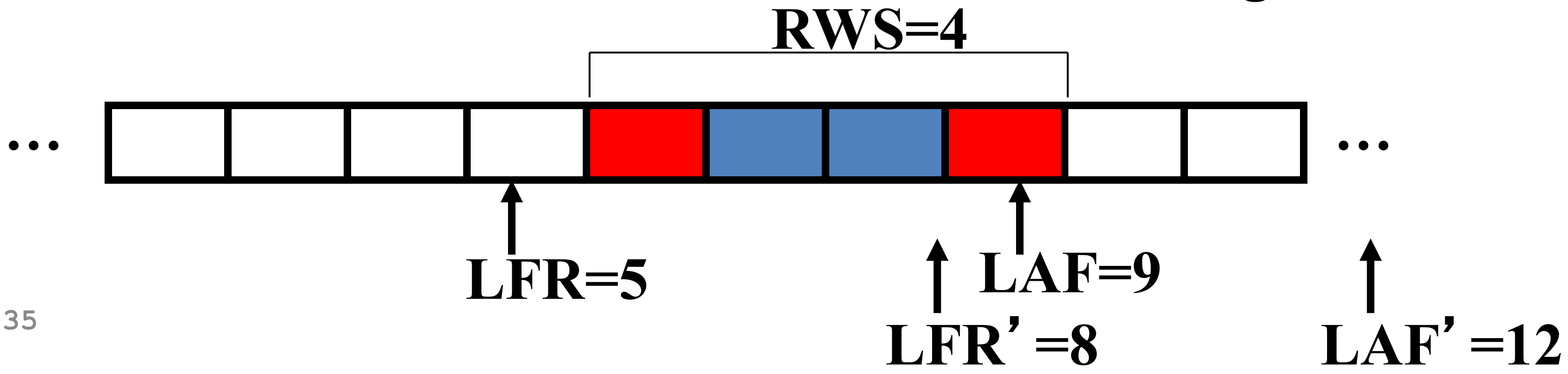
Selective Repeat Protocol (SRP)

- Selective ACK's sent by receiver identify specifically which frame(s) have been received correctly
  - In our example, the ACK would contain info that only packets #7 and #8 have already arrived at receiver
  - Sender only retransmits packets #6 and #9, and avoids resending #7 and #8
  - "sliding" window in SRP actually becomes much more complicated than GBN – track "holes" between acknowledged packets



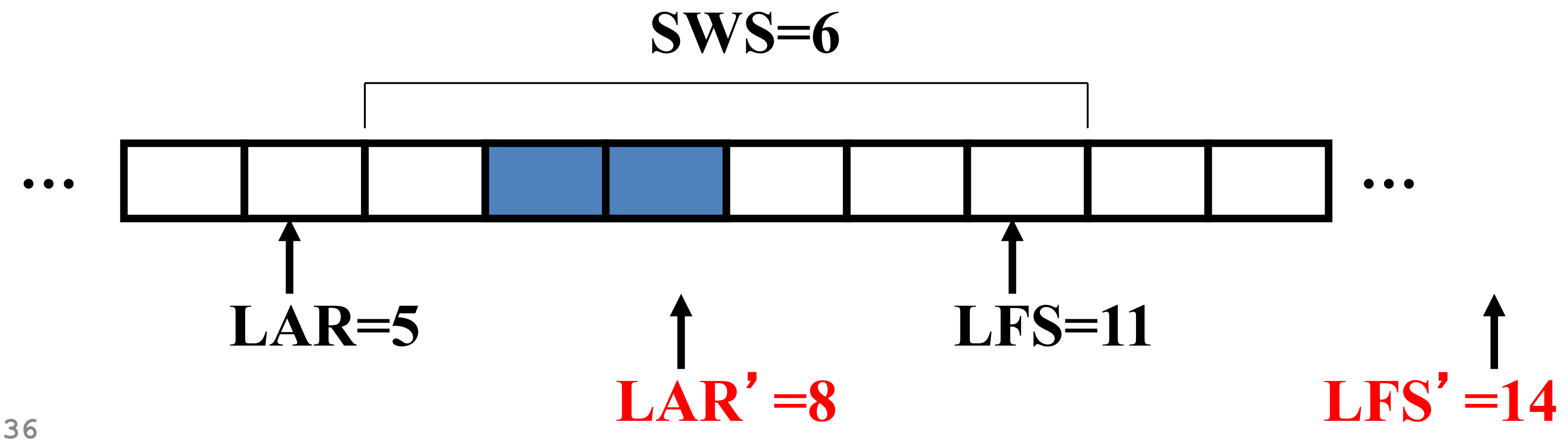
Selective Repeat Protocol (2)

- Example: RWS = 4, LFR = 5, LAF = 9
  - Suppose at receiver frames #7 and #8 arrive, but frames #6 or #9 either arrive out of order or are lost. Let's keep the data frames #7 and #8
  - When frame #7 arrives out of order, then send selective ACK with sequence #7
  - When frame #8 arrives, send selective ACK with sequence #8 (could selectively ACK whole window)
  - When frame #6 arrives, slide window right



Selective Repeat Protocol (3)

- Meanwhile, back at the sender…
  - Example: suppose SWS=6, LAR=5, LFS=11
  - When selective ACK #7 arrives, record that this packet was successfully received. When selective #8 arrives, record its successful reception. Don't have to retransmit packets #7 or #8
  - When ACK #6 arrives, slide window right



Link Layer vs. End-to-End Retransmission

- Can retransmit end-to-end (at transport layer) as well as at data-link layer
  - Yes, end-to-end retransmission over a multi-hop path while data-link layer retransmission is over a direct link
- Given link layer retransmission, why do you need end-to-end retransmission?
  - Packets can still be lost in intermediate nodes, e.g., routers
- Given end-to-end retransmission, why retransmit at link-layer at all?
  - Performance – don’t wait for end-to-end timeout, instead quickly retransmit on local link