Chapter 2
Efficient Reliable Protocol Design: Sliding Windows, GBN, SRP

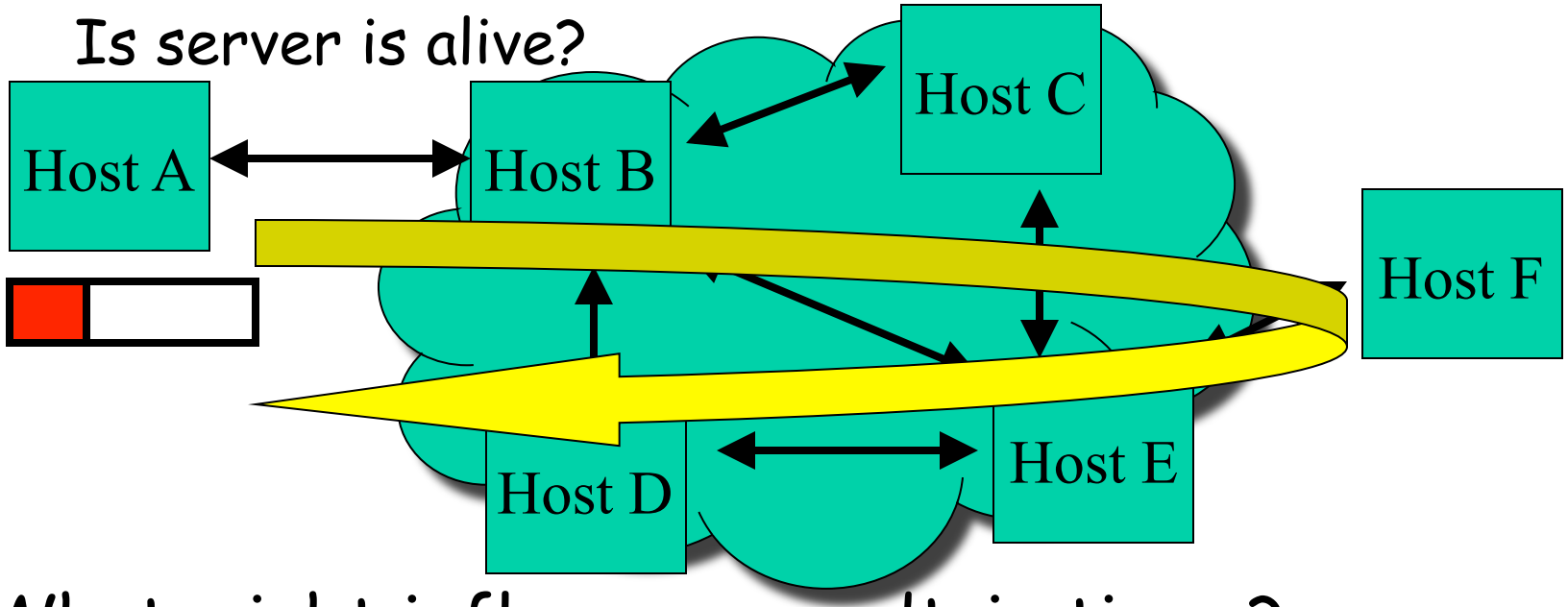
Prof. Rick Han
University of Colorado at Boulder
rhan@cs.colorado.edu

Announcements

- PA1: Show up for your time slots on time
- PS #1 is due Friday Sept 13 by 11:55 pm
- Next, Chapter 2, Efficient Reliable protocols

Problem Set #1: "Ping"

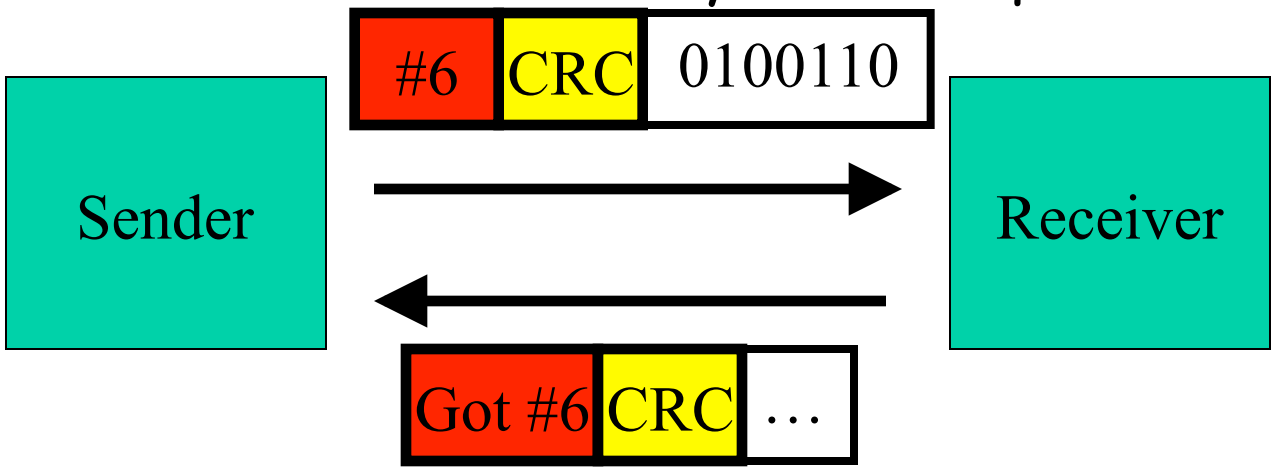
- "Ping" program allows you to send a packet to a host and have it echoed back
  - Used to probe the network for roundtrip times & packet losses
  - Is server is alive?



- What might influence roundtrip times?
  - Network delays (bandwidth, propagation, congestion), server delays, time of day, location, …

Recap of Previous Lecture

- Framing
  - Sentinel flags, byte/bit stuffing, and length fields
- Error detection – checksum, CRC
- Error correction - FEC
- Designing reliable ARQ protocols
  - Acknowledgements
  - Sequence numbers
  - Timeouts – choose wisely, about equal to RTT

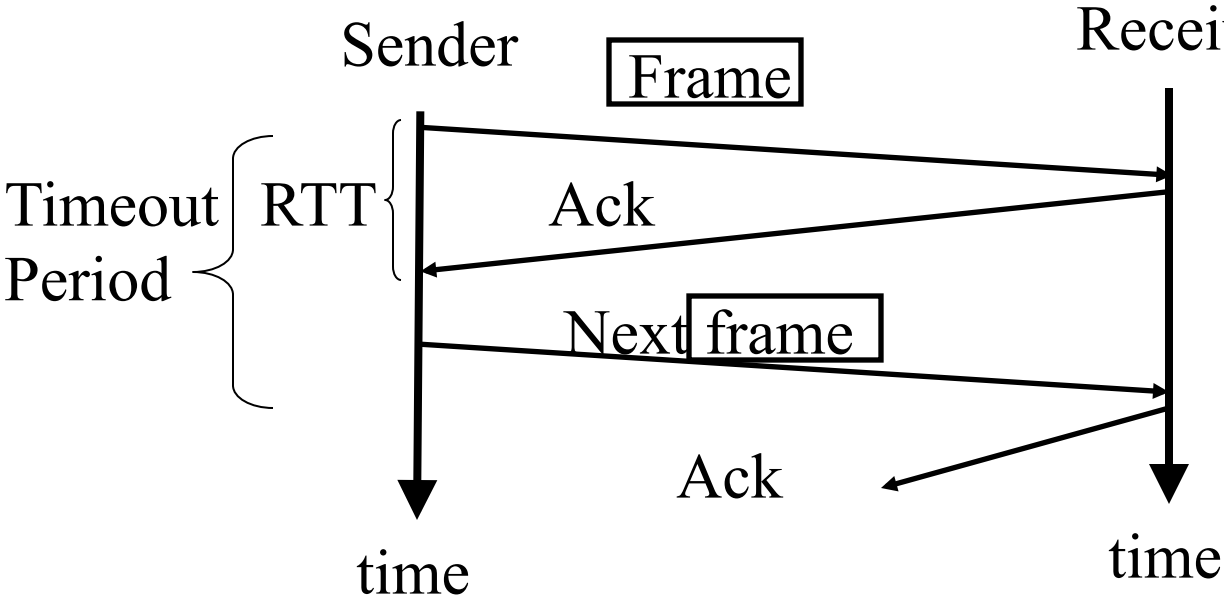


Designing *Efficient* Reliable Protocols

- Already seen one way to improve efficiency: choose timeout wisely
- Another way to improve efficiency: "keep the pipe full" with new data packets and necessary retransmissions
  - Stop-and-Wait
  - Go-Back-N
  - Selective Repeat (SRP)

Stop-and-Wait Protocol

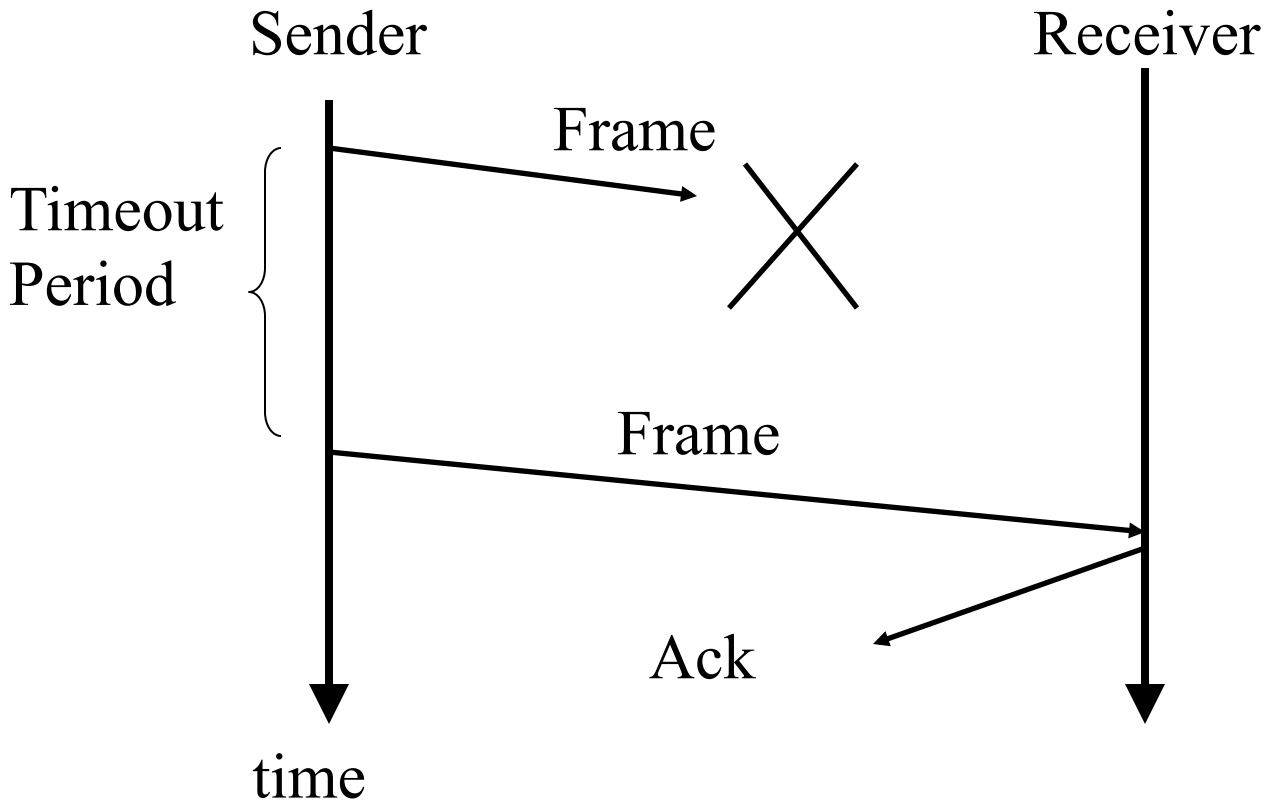
- After transmitting a packet/frame, the sender *stops and waits* for an ACK before transmitting the next frame
- If a timeout occurs before receiving an ACK, the sender retransmits the frame



- Link layer RTT = send time + process time at receiver + ACK send time + 2*(Prop delay)
- Here, timeout > RTT

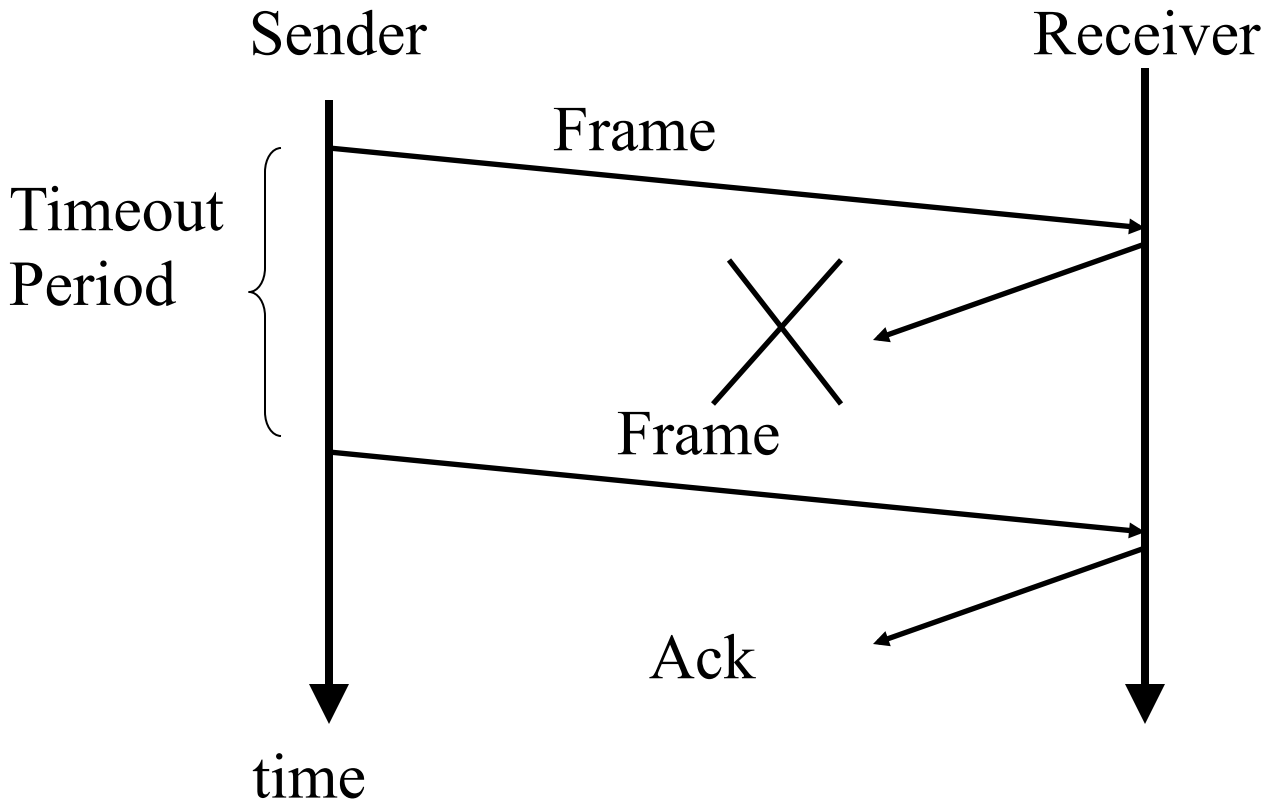
Stop-and-Wait Protocol (2)

- If a timeout occurs before receiving an ACK, the sender retransmits the frame.



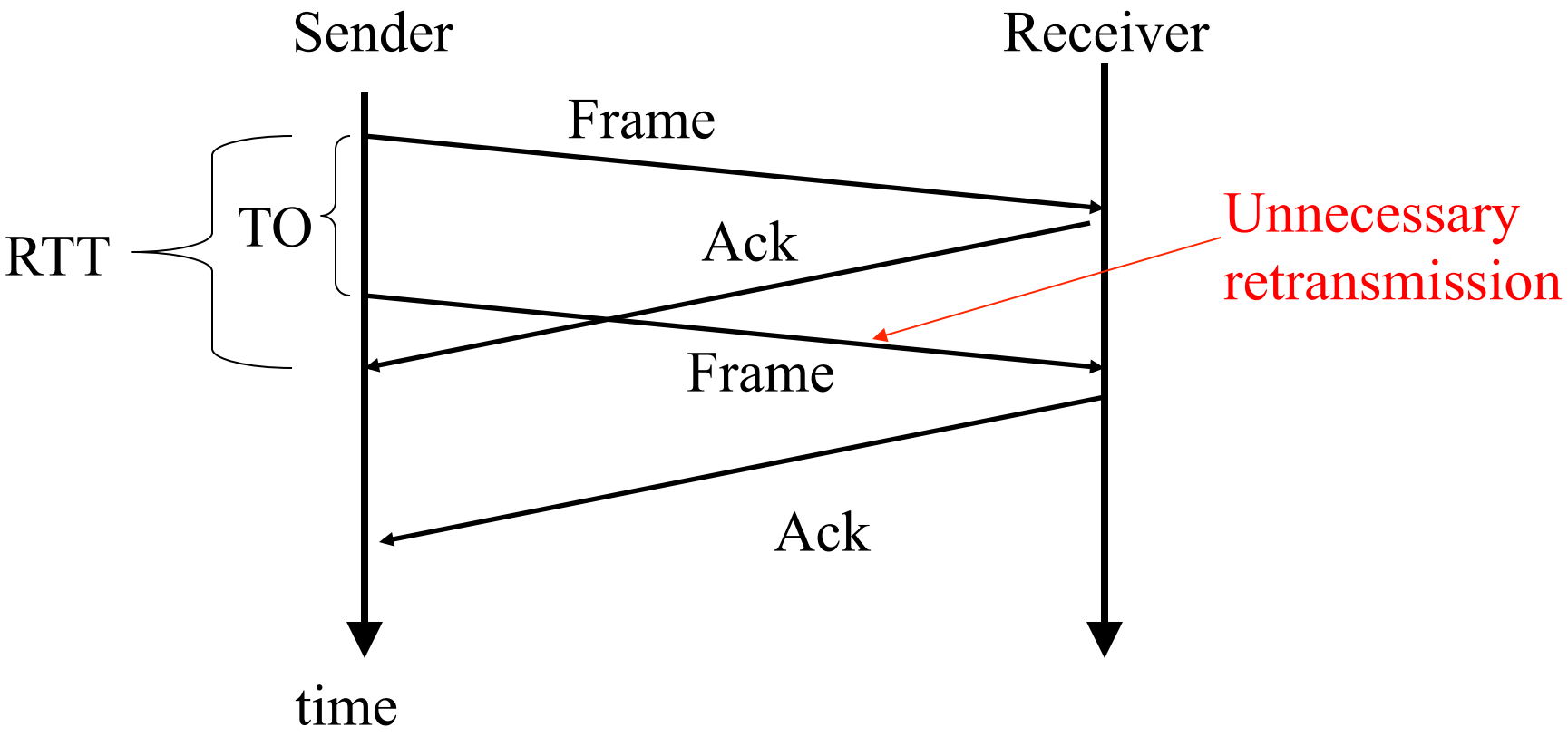
Stop-and-Wait Protocol (3)

- If a timeout occurs before receiving an ACK, the sender retransmits the frame.



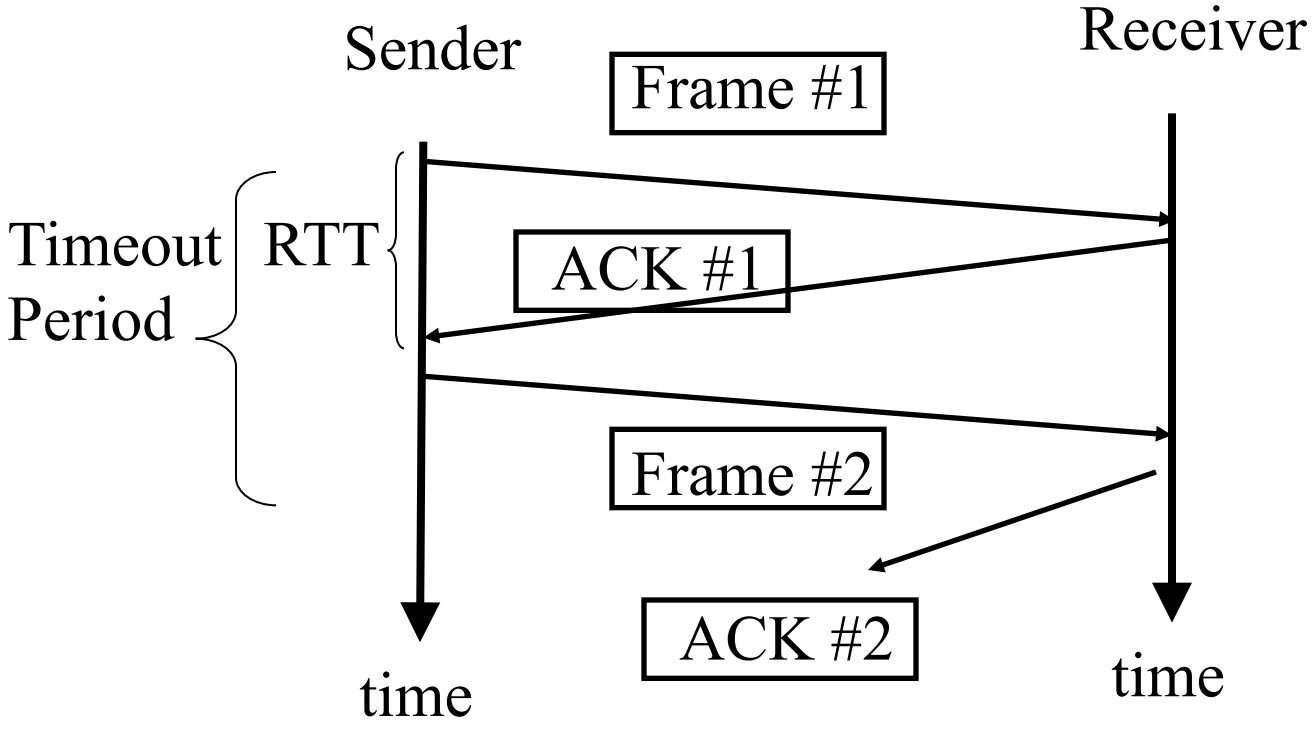
Stop-and-Wait Protocol (4)

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



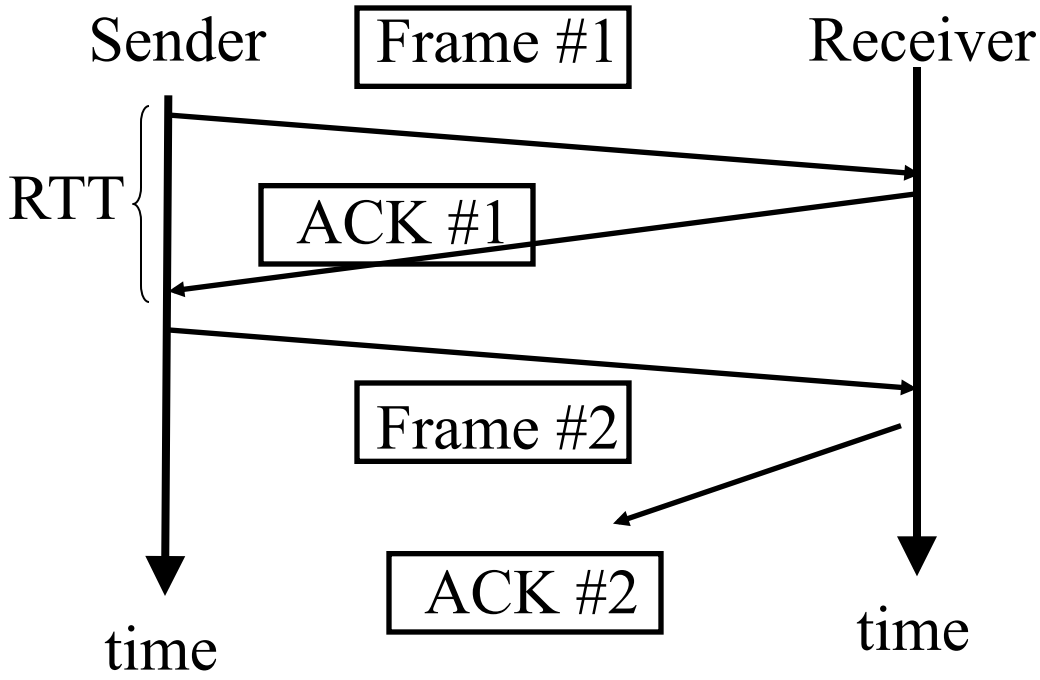
Stop-and-Wait Protocol (5)

- Label each packet and ACK with the proper sequence # to avoid confusion at receiver and sender



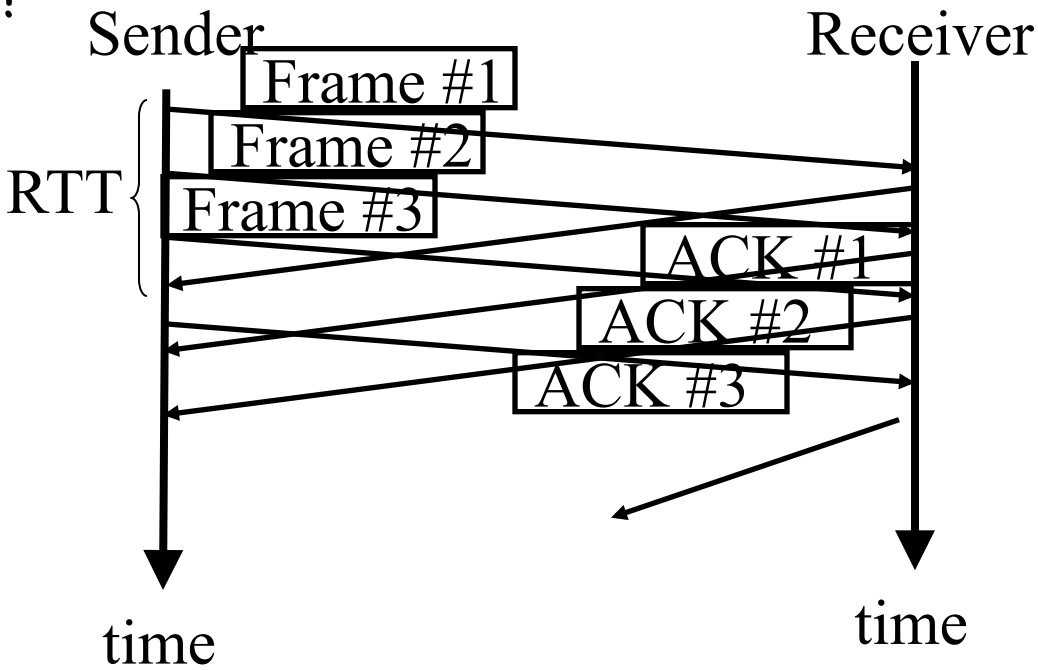
Problem with Stop-and-Wait

- Only one outstanding packet at a time => waste of link bandwidth
  - Example: 1.5 Mbps link with RTT 45 ms => a *Delay*Bandwidth product* = 67.5 Kb ≈ 8 KB. "pipe size"
  - If frame size = 1 KB, then use only 1/8 of bandwidth



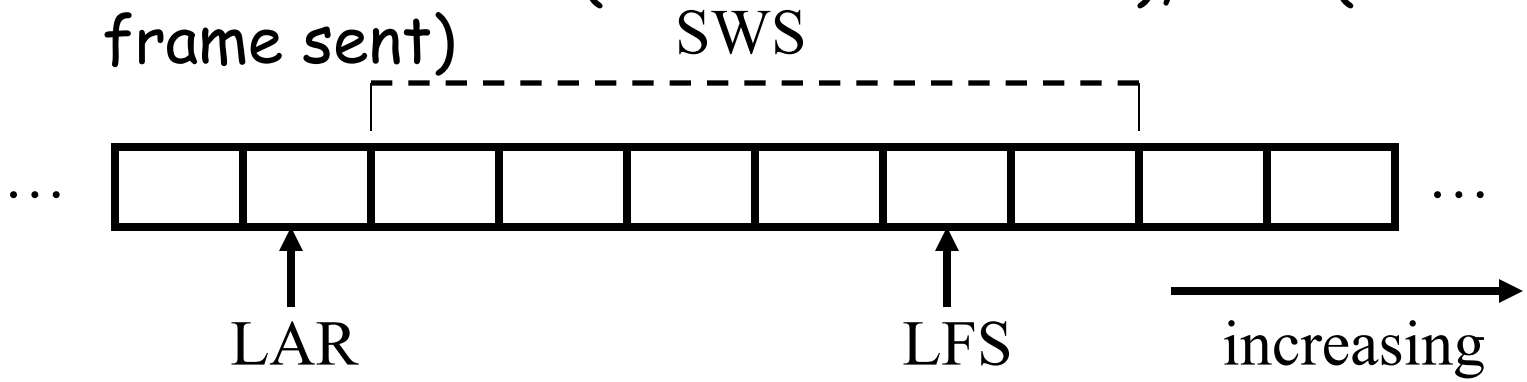
"Keep the Pipe Full"

- During one RTT, send N packets
  - Example: 1.5 Mbps link with RTT 45 ms => a *Delay*Bandwidth product* = 67.5 Kb ≈ 8 KB. "pipe size"
  - If frame size = 1 KB, and N=3, then can have 3 outstanding packets, 3/8 of BW, and triple bandwidth utilization!



Go-Back-N Sliding Window Protocol

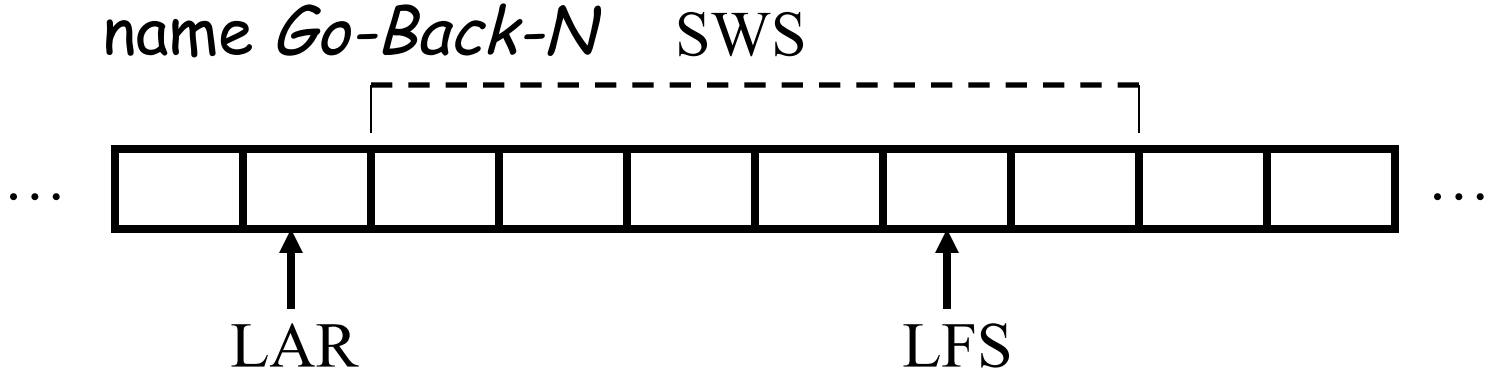
- Maintain a *sliding window* at both sender and receiver of unacknowledged packets
  - Send Window Size (SWS)
  - At sender: LAR (last ACK received), LFS (last frame sent)



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

Go-Back-N Sliding Window Protocol (2)

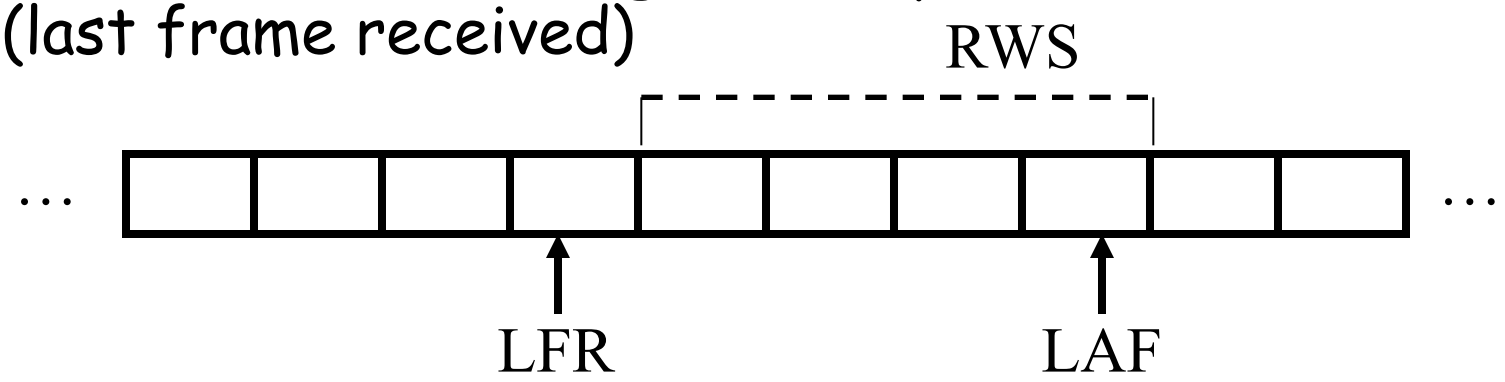
- Timeouts and retransmissions
  - Each frame in the window has its own timeout
  - When the timeout for frame #(LAR+1) expires, the entire window is retransmitted, hence the name *Go-Back-N*



  - 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 timer expires is accurate

Go-Back-N Sliding Window Protocol (3)

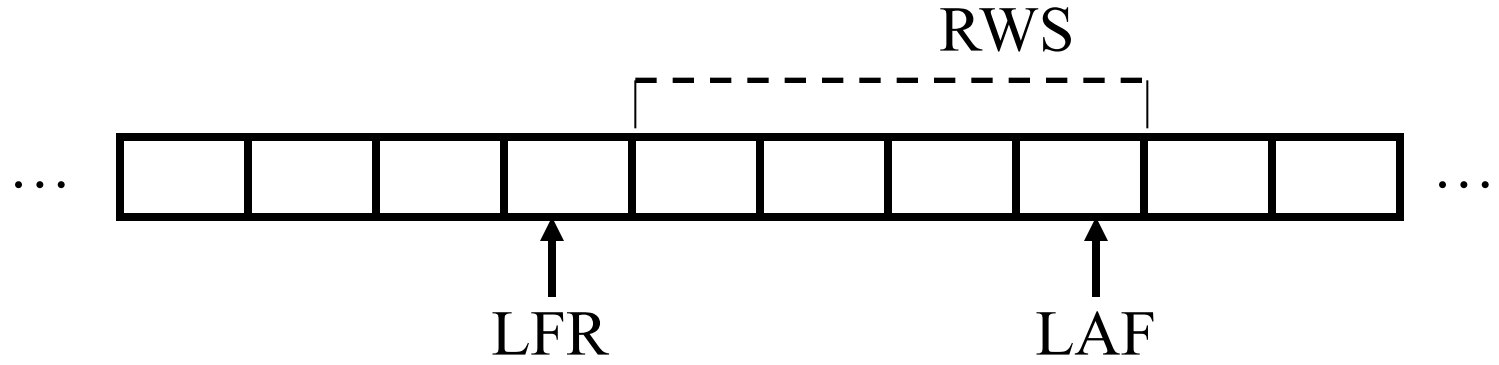
- At receiver:
  - Maintain a Receive Window Size (RWS)
  - At receiver: LAF (largest acceptable frame), LFR (last frame received)



- Sliding window:
  - When frame arrives, keep it if it's 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)

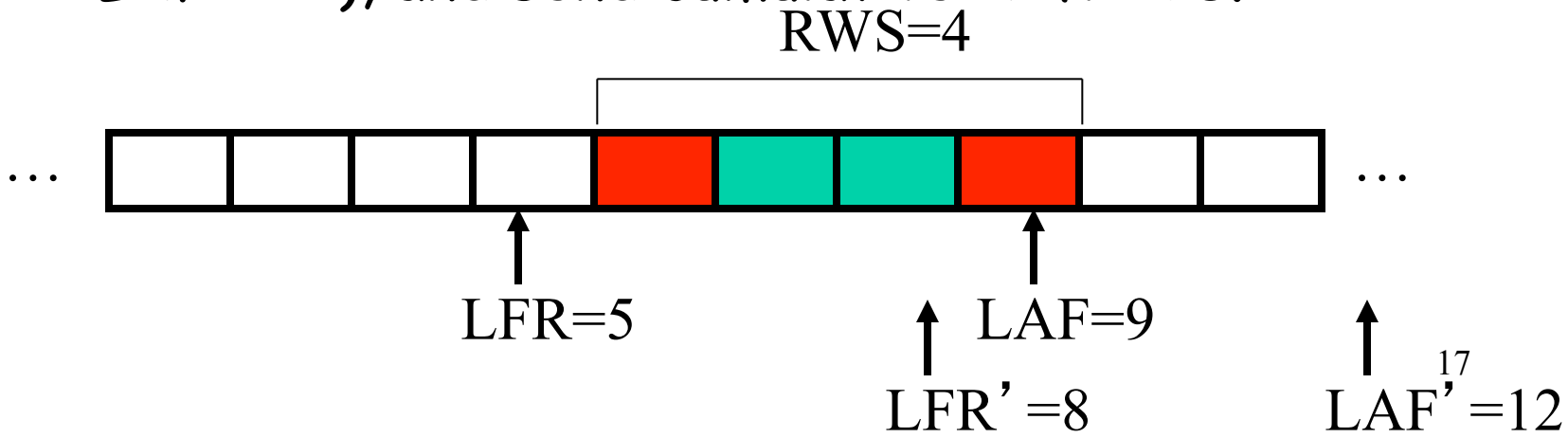
- A cumulative ACK = LFR says that everything up to the acknowledged sequence number has been received



- Simple to implement
- If you receive an out-of-order packet, you can
  - Data: keep or drop it
  - ACK: cumulatively ACK each time an (out of order) packet arrives (*advantage?*), 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

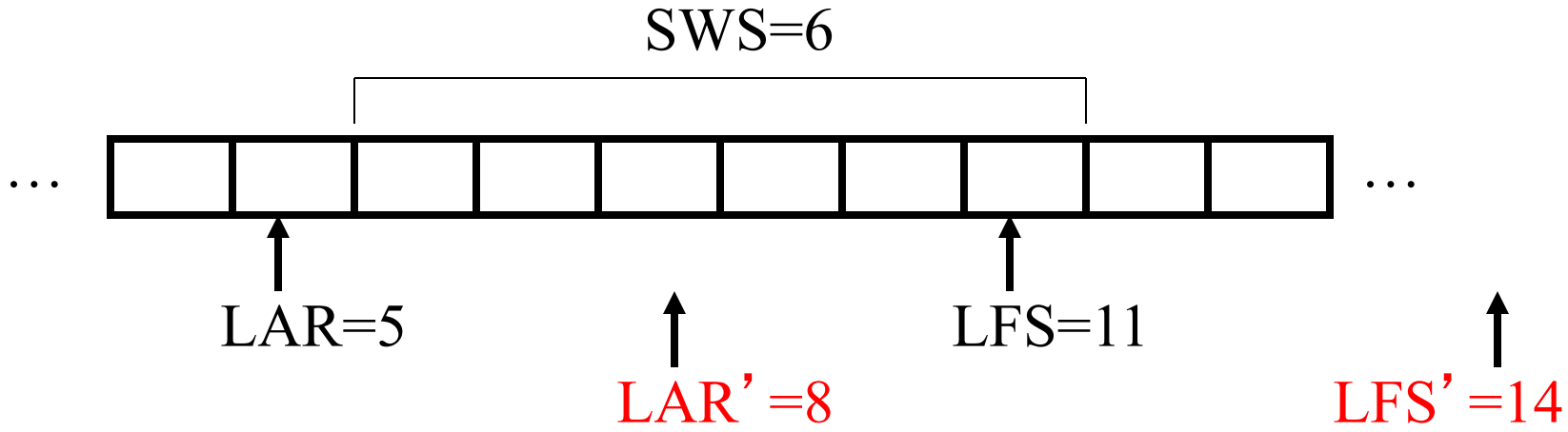
Go-Back-N Sliding Window Protocol (5)

- 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) (alternative: delay cumulative ACKs until #6 arrives)
  - 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)



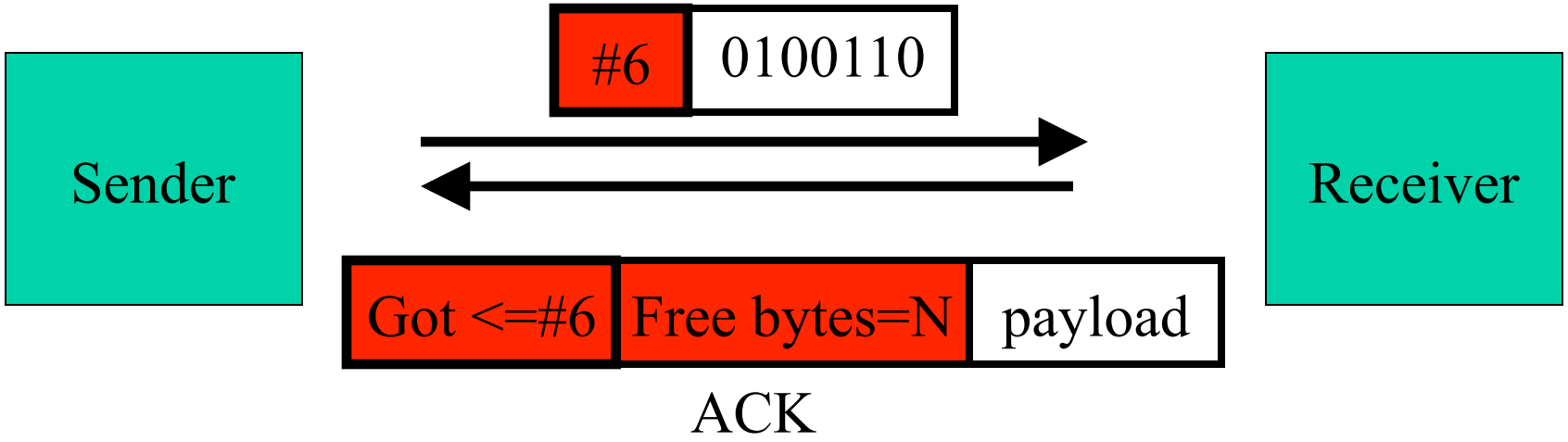
Go-Back-N Sliding Window Protocol (7)

- In previous example, SWS=6 <> RWS=4
  - Sender's effective window size is 4!
  - In general, effective window size is minimum of SWS & RWS
- What is the optimal window size?
  - = delay*BW product (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

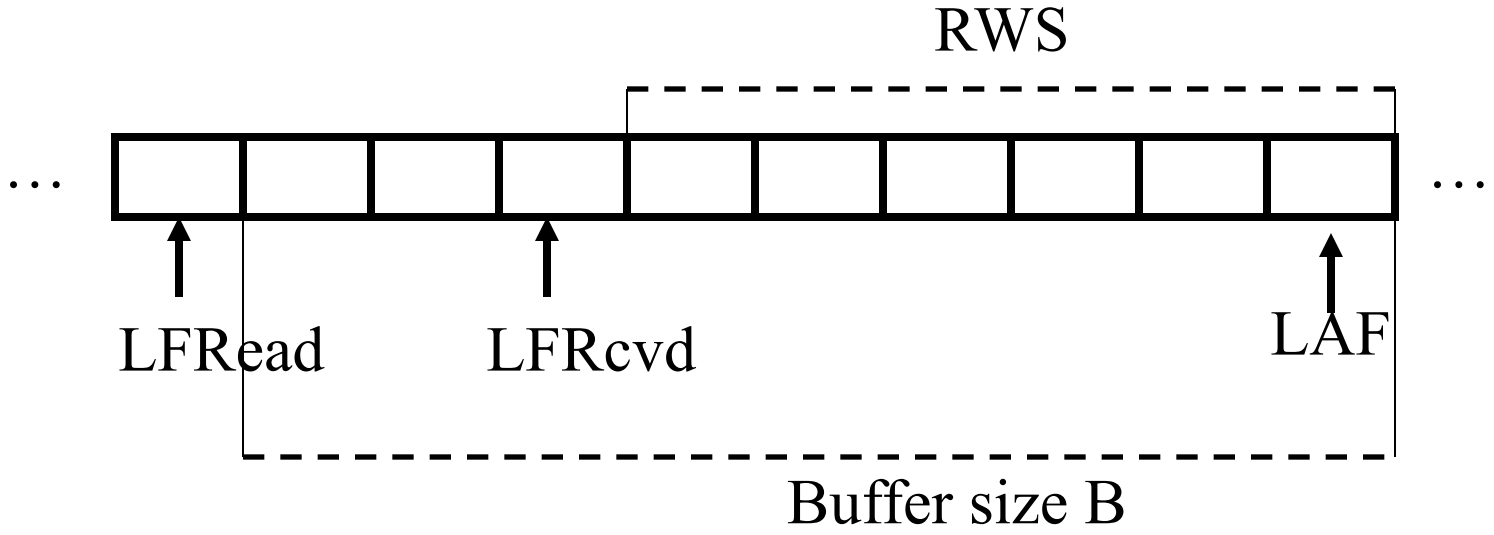
- Flow control deals with mismatch in rates between sender and receiver
  - If receiver is slow, then sender should slow down to match the receiver
- Two forms of flow control:
  - Rate-based flow control
    - If receiver can only process packets at a rate of *X* bits/sec, then construct a filter at 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)



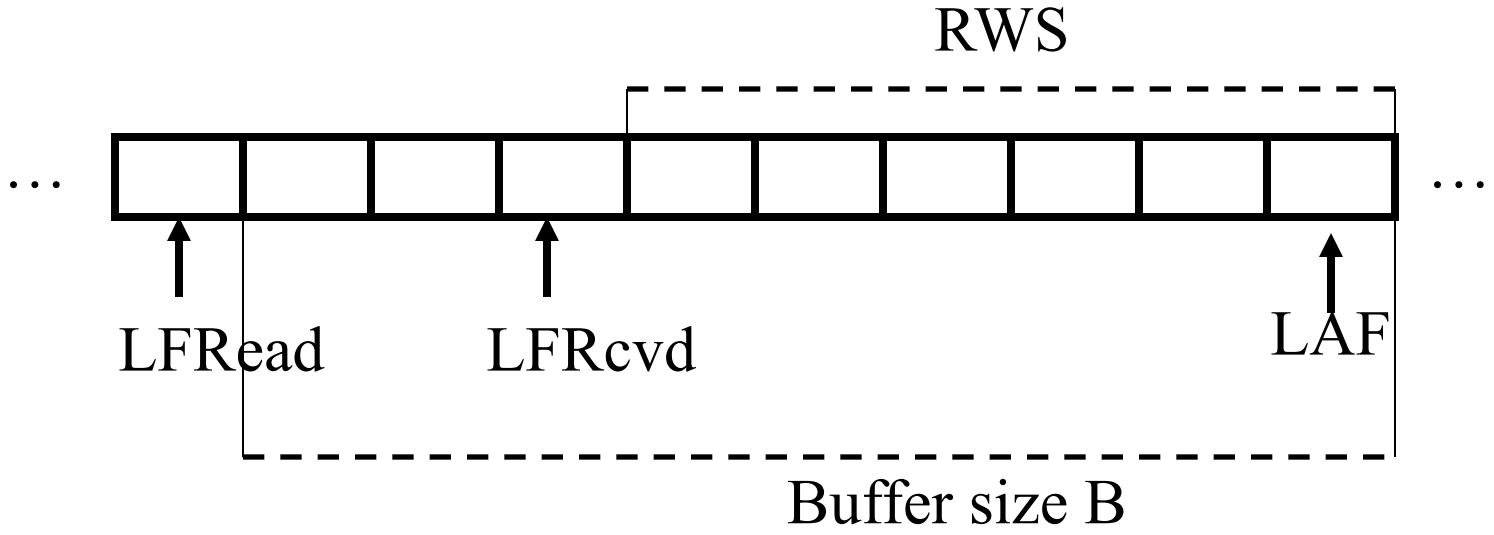
- Receiver allocates a finite buffer size B to receive data
  - RWS <= B
  - Application is slow to read data from buffer, so let LFRead = last frame read by application from buffer
  - To avoid confusion, rewrite LFR as LFRcvd

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 receiver 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)

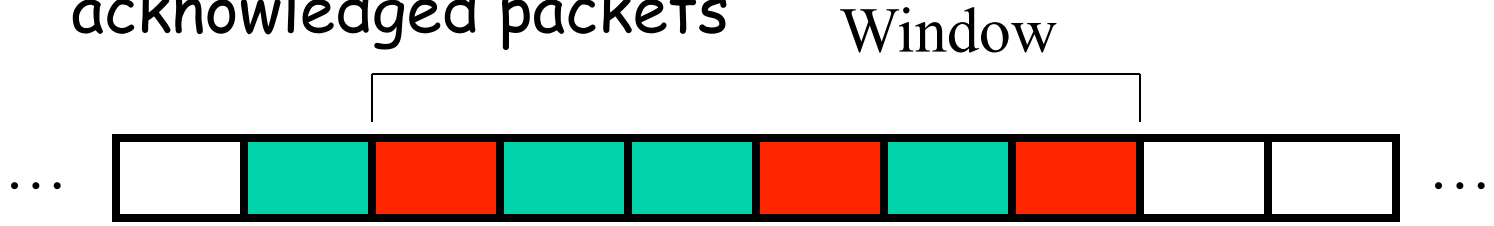
- Solutions to flow control deadlock:
  - when RWS=0, Receiver *periodically* generates an ACK if the free space window opens (Application reads data & frees up space) until new data arrives (must be periodic because a single ACK could be lost)
  - when RWS=0, Sender generates periodic probe data packets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

Problem with Go-Back-N

- Why are cumulative ACK's considered inefficient?
  - Cumulative ACK's cause 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

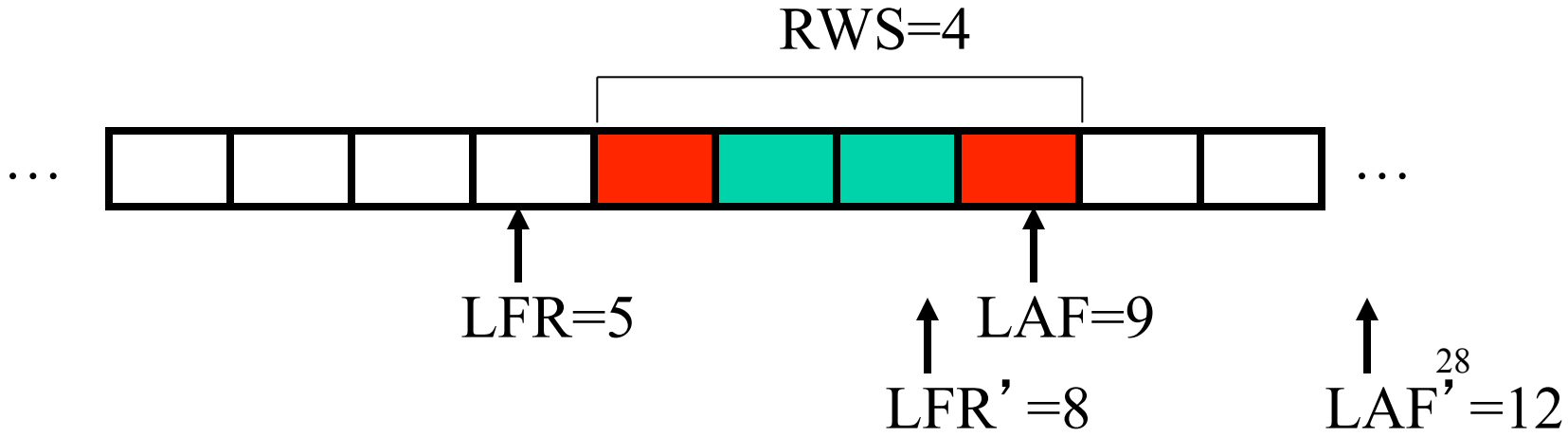
Selective Repeat Protocol (SRP)

- 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" btwn. acknowledged packets



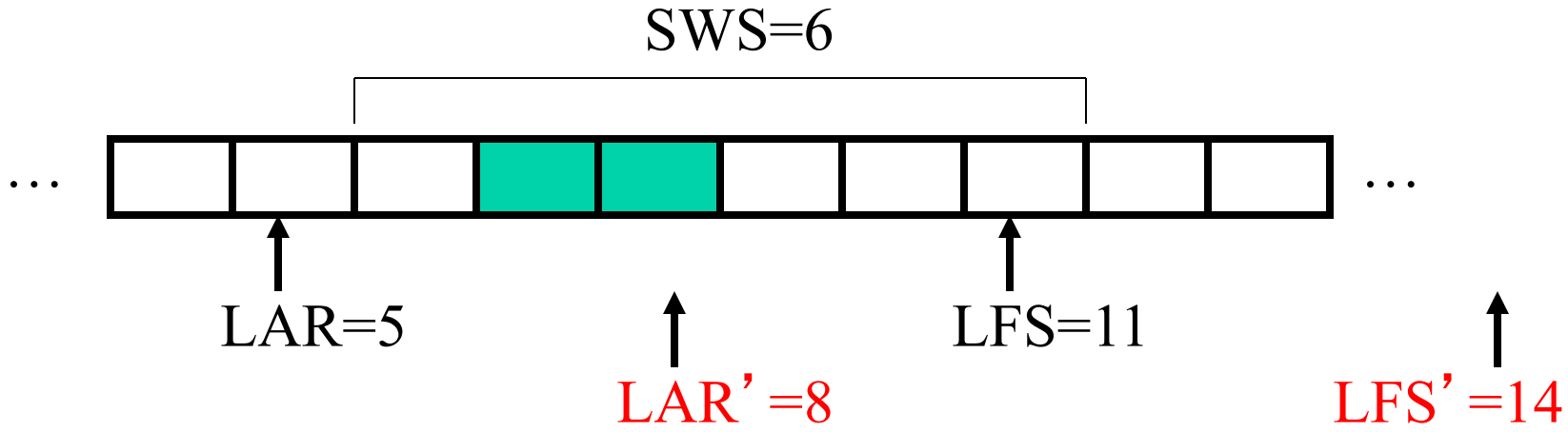
SRP (2)

- 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)



SRP (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.



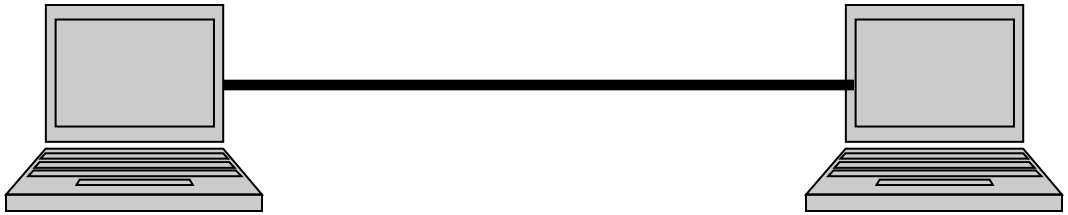
Other Protocol Design Issues

- Congestion control (will cover in later chapter)
- Finite sequence # wrap-around (see text)
- Connection establishment & close use state machines (will cover in later chapter)
- Full duplex vs. half-duplex – usage of terms
  - Interpretation #1 (analog channel):
    - Full duplex = can send data in both directions simultaneously, e.g. over two different frequencies
    - Half-duplex = can send data in both directions, but not simultaneously, e.g. communication over shared medium
    - Simplex = can send data in only one direction
  - Interpretation #2 (digital protocols):
    - Full duplex = piggybacking data with an ACK
    - Half-duplex/simplex = don't piggyback data with an ACK

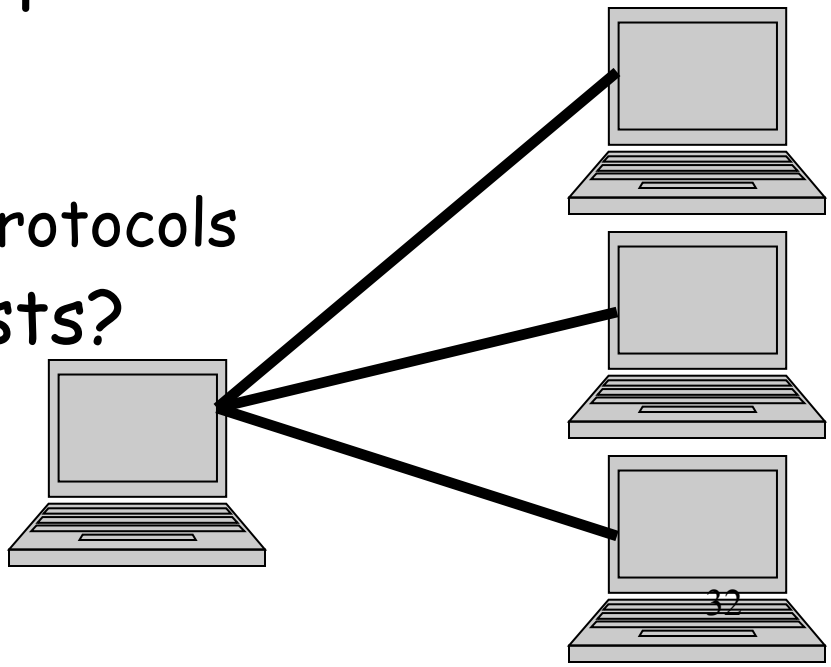
Link Layer vs. End-to-End Retransmission

- Can retransmit end-to-end (at transport layer) as well as at data-link layer
  - Retransmit over a multi-hop path instead of a single 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

Direct-Link or Point-to-Point Networks

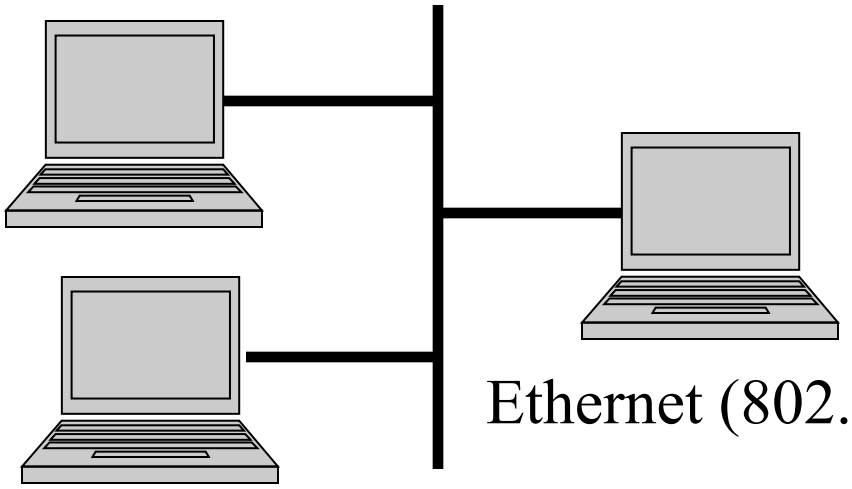


- Physical layer handles bits
- Data-link layer handles packets
  - Framing
  - Error detection
  - Retransmission-based protocols
- How do I send to N hosts?
  - N point-to-point links
  - Other possibilities?

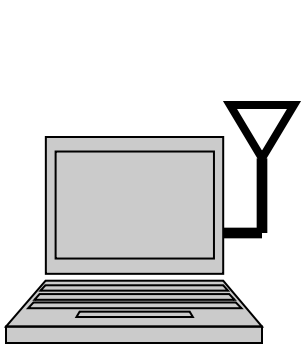


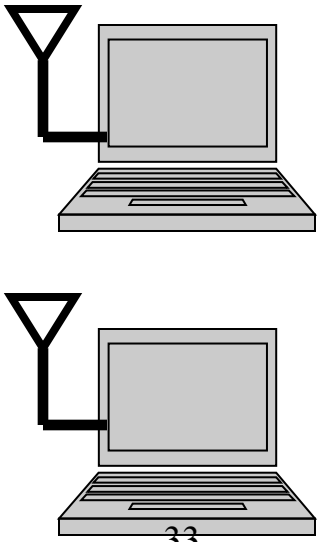
Shared-Media or Broadcast Networks

- N senders and receivers connected by a shared medium (copper wire, atmosphere)
- Sharing access to the same media
  - Analogy: How do N persons converse in a room or at the dinner table? At once, or one by one? What is the communications protocol?



Ethernet (802.3)

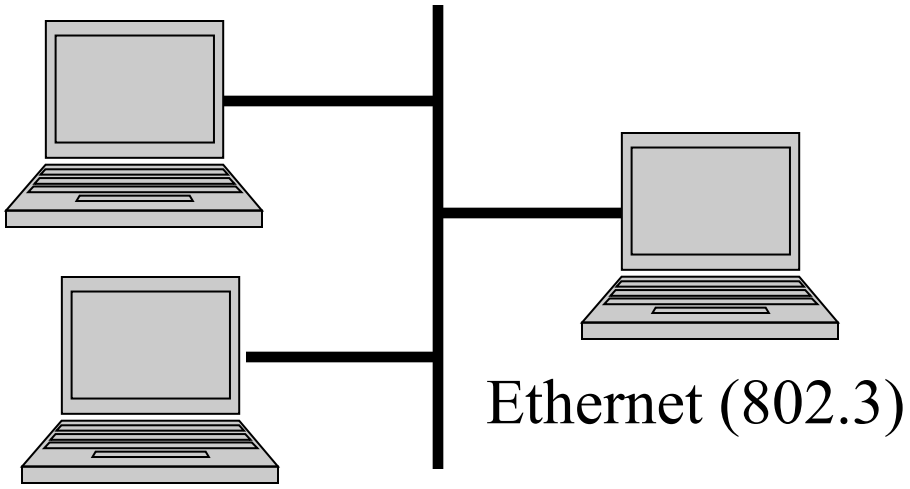




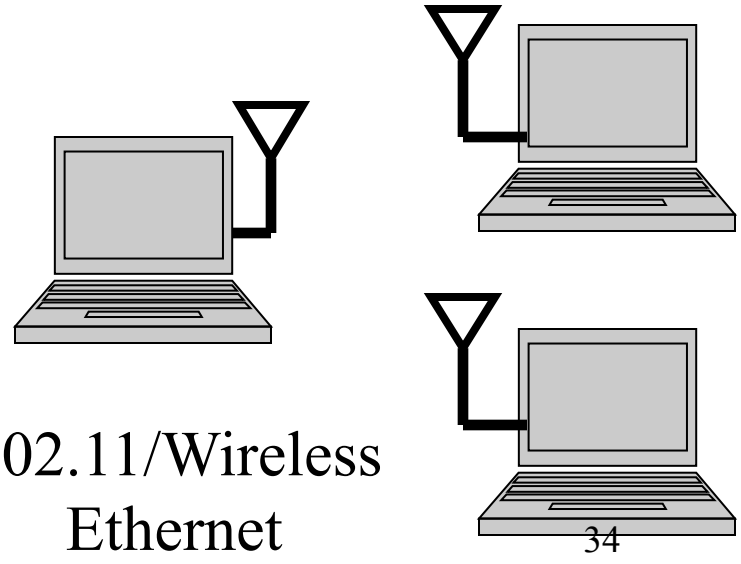
802.11/WiFi

Shared-Media or Broadcast Networks

- Local Area Network (LAN)
  - Ethernet, Fast Ethernet, Gigabit Ethernet, …
  - Wireless Ethernet, or 802.11 a/b/g/n/ac/ad, or Wireless Fidelity (WiFi)



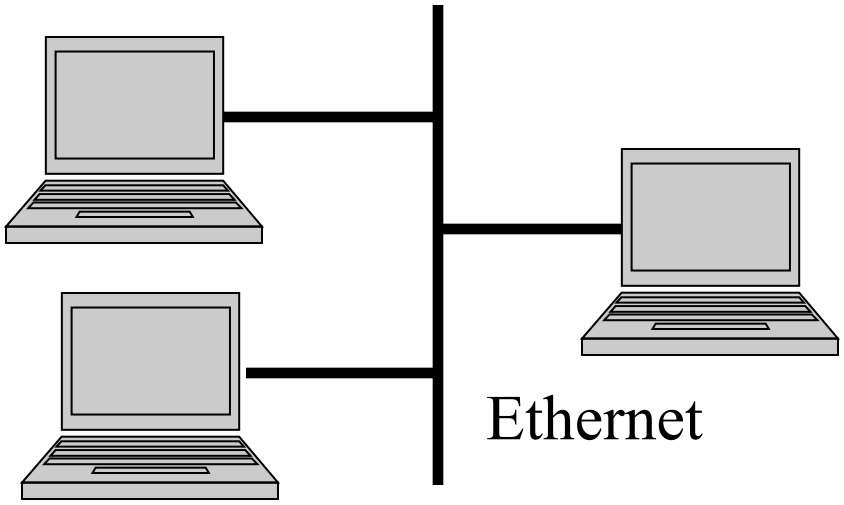
Ethernet (802.3)

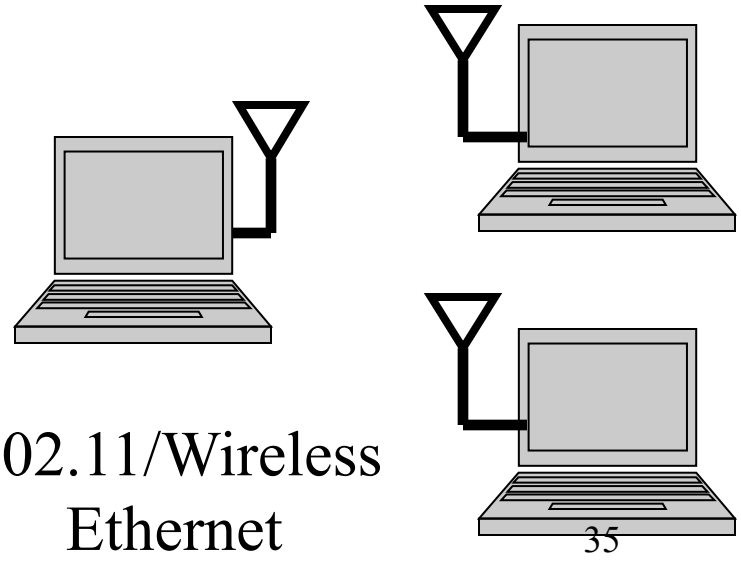


802.11/Wireless Ethernet

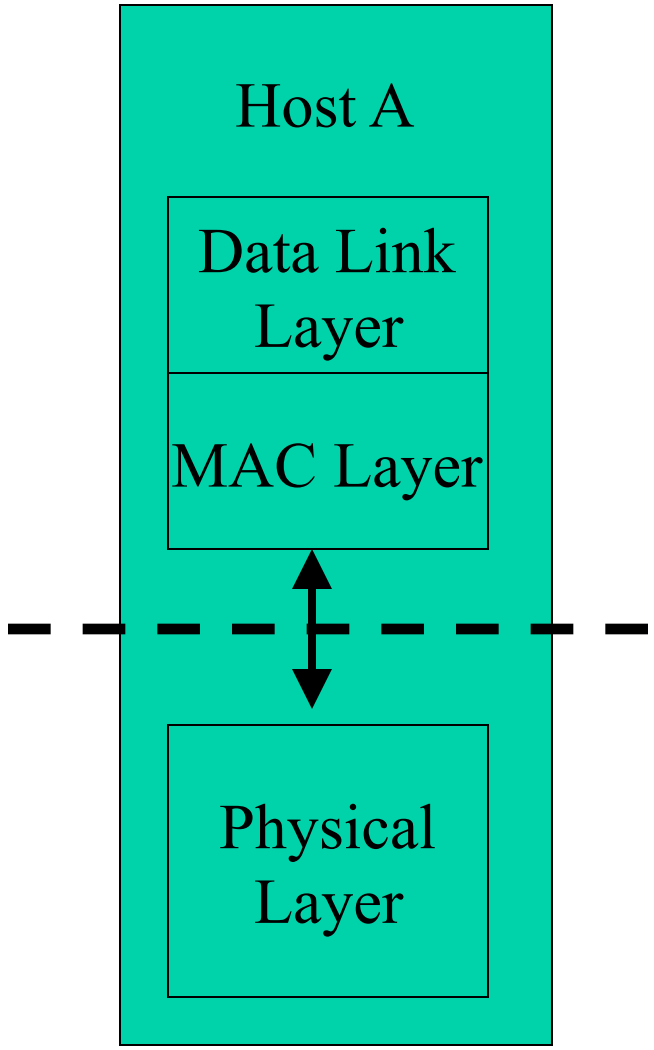
Multiple Access Protocols

- Determine which host is allowed to transmit next to a shared medium
  - Channel reservation: TDMA, FDMA, CDMA, Token Ring, …
  - Random access: ALOHA, CSMA/CD, CSMA/CA





Multiple Access Protocols (2)



- Also called Medium-Access Control (MAC) protocols
- Before data link-layer packets can be sent, e.g. GBN/SRP, a sender has to gain access to the media first
  - MAC layer is often placed in the stack between layer 2 and layer 1