ECE 650
Systems Programming & Engineering

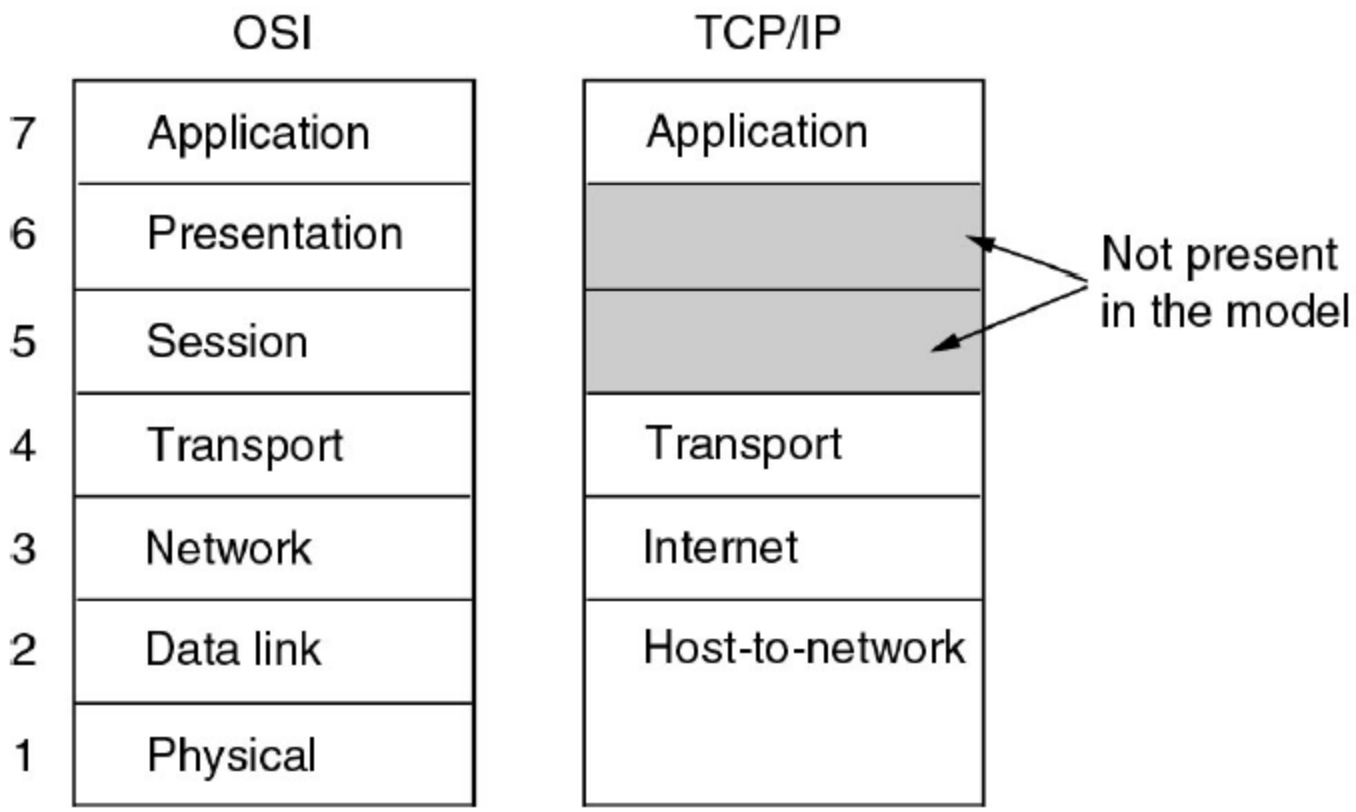
Spring 2018

Networking – Link Layer

Tyler Bletsch
Duke University

Slides are adapted from Brian Rogers (Duke)

TCP/IP Model


OSI
TCP/IP
7
Application
6
Presentation
5
Session
4
Transport
3
Network
2
Data link
1
Physical
Application
Transport
Internet
Host-to-network
Not present
in the model

Layer 1 & 2

- Layer 1: Physical Layer
  - Encoding of bits to send over a single physical link
- Layer 2: Link Layer
  - Framing and transmission of a collection of bits into individual messages sent across a single subnetwork (one physical topology)
  - Provides local addressing (also known as Media Access Control (MAC))
  - May involve multiple physical links
  - Often the technology supports broadcast: every "node" connected to the subnet receives
  - Examples:
    - Modern Ethernet
    - WiFi (802.11a/b/g/n/etc)

Physical Layer

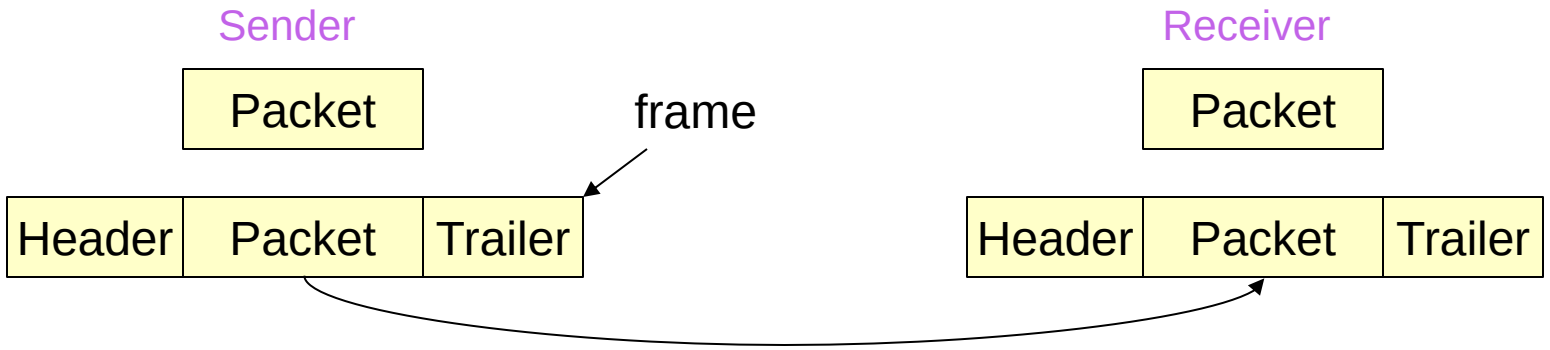
- We are not going to cover this in detail in this course
- But to give you some idea of what this covers:
  - Fourier analysis
  - Channel data rates
  - Transmission media:
    - Guided: twisted pair, coaxial cable, fiber cables
    - Wireless: electromagnetic waves, radio transmission, microwaves, infrared, lightwave
    - Communication satellites
  - Modulation / Demodulation
  - Frequency division and time division multiplexing
  - Packet switching vs. circuit switching
  - GSM, CDMA

Link Layer

- Algorithms for communication between adjacent machines
  - Communication that is both
    - Reliable
    - Fast
  - Adjacent machines means directly connected by a comm channel
    - E.g. coaxial cable, telephone line, point-to-point wireless channel
    - Channel is "wire-like" in that bits delivered in same order they are sent
- Seems like a simple problem, but…
  - Errors in bit transmissions can happen
  - Finite data rates & bit propagation delays have impact on efficiency
- Link layer protocols handle these aspects

Link Layer - Framing

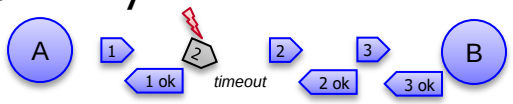
- Link layer functions
  - Provide service interface to the network layer (next layer up)
  - Handle errors during transmission
  - Regulate data flow (so receivers not flooded by faster senders)
- Basic mechanism is "framing"
  - Receive packets down from network layer
  - Break the packet up into frames; add header and trailer
  - Frames sent across transmission medium



Service to Network Layer

- Service: transfer data provided by network layer on one machine to the network layer on another machine
  - Of course the actual data path travels across physical layer
- Some common types of link layer services
  - Unacknowledged connectionless service
    - Send frames from src to dest where dest does not acknowledge
    - If frame is lost due to noise, no attempt to detect or recover
    - Useful for low error rate channels or real-time traffic
      - e.g. voice where late data is worse than bad data

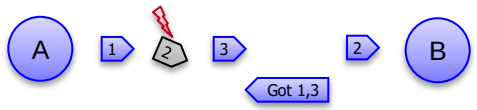
  - Acknowledged connectionless service
    - Still no connection established, but dest acknowledges each frame
    - Sender can re-send frames not ack'ed within a time limit
    - Link layer ack is an optimization, never requirement – why?
    - Good for highly unreliable channels (e.g. wireless)



  - Acknowledged connection-oriented service
    - More on next slide

Acknowledged Connection-Oriented

- Most sophisticated service
  - Src and dest machine establish a connection
  - Each frame sent over connection is numbered and ack'ed by dest
  - Guarantees each frame is received exactly once and in the proper order
    - Connectionless service could cause a packet to be sent several times (if acks are lost)
- 3 phased approach
  - Connection is established
    - Each side initializes variables & counters to track frames send & received
  - One or more frames are transmitted
  - Connection is released: free up variables, buffers, other resources

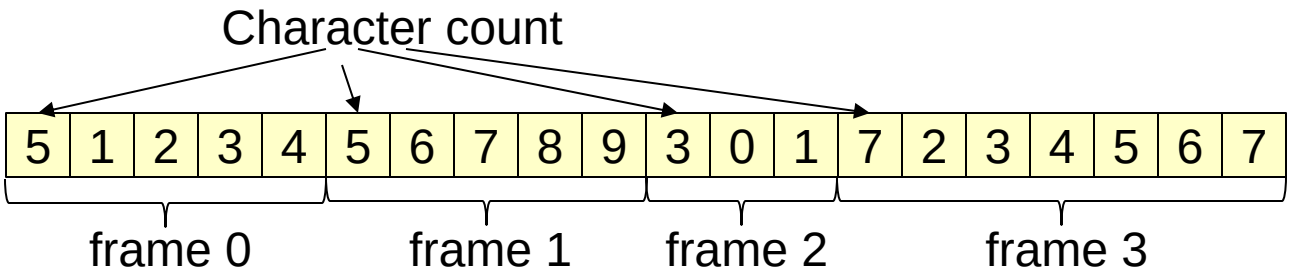


Framing

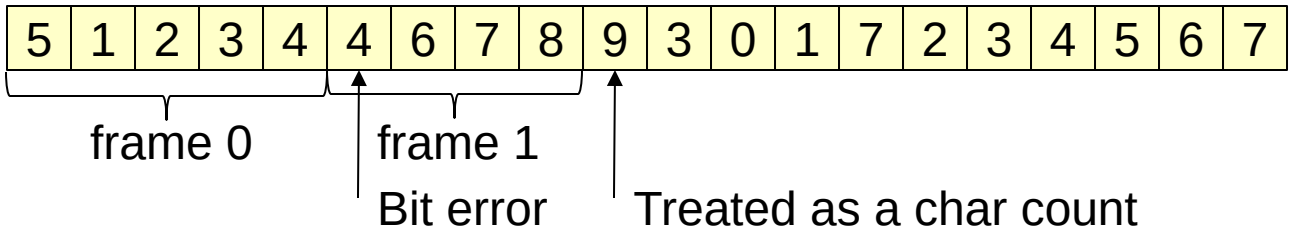
- Link layer sends stream of bits across physical layer
  - On unreliable links, bits may be lost or altered values
  - Link layer can detect or correct bit errors
- Error handling is simplified by "framing"
  - Break up bit stream into frames
  - Add some sort of checksum to each frame
  - Receiver recomputes checksum from received frame bits
  - If checksums do not match, then recovery happens
    - E.g. correct the error or send a negative ack to sender to resend frame
- How does link layer divide a bit stream into frames?
  - More complex than it may initially seem

Character Count

- Think of the bit stream as a sequence of characters
- Add 1 new character to the start of each frame
  - Value indicates the # of characters in this frame



- Transmission errors make this difficult to use in practice
  - Bit errors in the character count cause the destination to become out of sync with the sender; no longer knows where frames are



Flag Bytes with Byte Stuffing

- Denote start and end of each frame with special bytes
  - Called a "flag byte"
  - Two consecutive flag bytes indicates the end of one frame and start of the next.
- If a receiver loses synchronization
  - Simply search for the flag byte to find end of the current frame
  - Restart processing frames at the next one
- What if frame includes a bit sequence that matches flag?
  - Link layer inserts another special byte before that sequence
  - Called an "escape byte"
  - Receiving link layer SW strips out escape & flag bytes
- Disadvantage: Tied to a fixed char size (e.g. 8B or 16B)

Bit Stuffing w/ Start & End Flags

- Allows flags with arbitrary # of bits
- Each frame begins & ends with bit pattern
  - Again, a flag: 01111110
- Bit stuffing on frame payload by sender:
  - When a sender sees 5 consecutive 1's, it stuffs a 0 in the stream
- Bit de-stuffing on frame payload by receiver:
  - When it sees 5 consecutive 1's followed by a 0, remove the 0 bit
- All fully transparent to network layer
- Boundary between frames can be unambiguously found

0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0	Frame payload
0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0	After bit stuffing
0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0	After bit de-stuffing

Error Control

- Several options:
  - Ignore errors in data link layer; let higher-layer in stack handle it
  - Send response frames with positive or negative acks
  - Combine positive & negative acks with timers
    - What happens if an entire frame is lost? Receiver will never ack
    - Set a timer on frame transmit; re-send if no ack before timer expires
  - Combine acks with timers and sequence numbers
    - What if ack is lost?
      - Receiver got frame, but sender will resent after timer expires
      - Receiver will receive multiple copies of same frame
    - Give each outgoing frame a sequence number
    - Receiver can distinguish re-transmissions from originals
    - Ensure each frame is processed by receiving data link layer only once

Flow Control

- What if sender can send faster than receiver can receive?
  - Eventually receiver would be flooded and begin to drop frames
- Typically in link layer, feedback-based flow control is used
  - A protocol with a set of rules
  - Rules define when a sender may transmit a new frame
  - For example, frames sent only when receiver gives permission
    - Receiver may say on connection setup, send up to N frames now
    - Sender waits until receiver tells it to send more
  - More details in a bit…

Error Detection vs. Correction

- Receiver link layer can detect or correct some bit errors
- Correction is more expensive than detection
  - In terms of # of overhead bits (checksum)
- Tradeoff:
  - On reliable channels (e.g. fiber), often use cheaper detection
  - On unreliable channels (e.g. wireless), often use correction

Hamming Distance

- **Hamming Distance** is the number of bit differences between two binary strings

- 011010
  011110
  Hamming distance: 1

- 011010
  101011
  Hamming distance: 3

- If two bit strings have a Hamming distance of ***d***, it would take ***d*** bit errors to turn one into the other.

Error Detection and Correction

- Let's consider a frame is now n = m + r bits
  - m = data bits (the message)
  - r = redundant bits (e.g. checksum)
  - Sometimes the n-bit chunk is called a **codeword**

- A simple error-detection code: Parity
  - Add a single redundant bit to the message
    - Even (odd) parity: add bit to make number of 1 bits even (odd)
    - Need Hamming distance 2 to fool it – any single-bit error produces codeword w/ wrong parity
    - Thus can detect any single bit error

Application of Hamming Distance

- For a message, usually all 2^m values are possible
  - But algorithms to compute check bits usually don't result in all 2^n codewords being used
    - In other words, not all codewords are *valid* (i.e., have **r** bits consistent with **m** bits)
  - Based on algorithm, we can enumerate all possible codewords
  - Hamming distance of the code is the minimum Hamming distance between any 2 valid codewords
- Error detection and correction properties
  - Requires a distance d+1 code to **detect** all d single-bit errors
    - No way for d single-bit errors to convert one valid code to another
  - Requires 2d+1 code to **correct** all d single-bit errors
    - Valid codewords are so far apart that with d single-bit errors, the original valid codeword is still the "closest"

Error Correction

- Example: Assume 4 valid codewords
  - 0000000000, 0000011111, 1111100000, 1111111111
- Hamming distance of 5
  - Can *detect* 4 single-bit errors
  - Can *correct* 2 single-bit errors
- Scenario: Receiver gets 0000000111
  - Most likely (closest distance): Original codeword was 0000011111
    - Two bit errors corrected: 0000000111
  - Possible but less likely: Original codeword was 0000000000
    - In this case, there were 3 bit errors: 0000000111
    - The error will not be properly corrected!

Error Correction – Some Principles

- What if we want to correct any single bit errors?
  - Each of 2^m messages has n illegal codewords at distance 1
    - I.e., each of the n bits could be flipped
  - Each message requires n+1 bit patterns dedicated to it
    - Correct codeword + n codewords at distance 1
  - As a result, we have the following inequality
    - $(n+1)\ 2^m \leq 2^n$
    - Substituting $n = m + r$:
      $(m + r + 1)\ 2^m \leq 2^m\ 2^r$
    - Cancel 2^m term:
      $(m + r + 1) \leq 2^r$
    - Given r, this shows how many message bits can be protected from a single-bit error by r redundancy bits:
      $m \leq 2^r - r - 1$
      - This at scale, this works out to *basically* $m \leq 2^r$

Error Correction Example: Hamming codes

- Let's go through a common example error correction code: **Hamming code**
  - Number codeword bits consecutively starting at 1 on left
  - Bits that are powers of 2 (1, 2, 4, …) are the check bits, rest of bits are data bits
  - Each check bit is the parity of some set of the bits
  - To see which check bits a data bit in position *k* contributes:
    - Rewrite *k* as a sum of powers of 2, e.g. 13 = 1 + 4 + 8
  - Coverage example:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	…
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

Figure from Wikipedia "Hamming code"

- Example: 1100101 -> 00111000101 (blue bits are the check bits)
- If check bits p2 and p8 both indicate error, then bit 2+8=10 is the cause (assuming a single bit error)

Error Detection

- Widespread practice is to use CRC
  - Cyclic Redundancy Check
- Goal: maximize protection, minimize bits
- Consider message to be a polynomial
  - Each bit is one coefficient
  - E.g., message 10101001 -> $m(x) = x^7 + x^5 + x^3 + 1$
- Can reduce one polynomial modulo another
  - Let $n(x) = m(x)x^3$. Let $C(x) = x^3 + x^2 + 1$.
  - n(x) "mod" C(x) : r(x)
    - Find q(x) and r(x) s.t. $n(x) = q(x)C(x) + r(x)$ and degree of r(x) < degree of C(x)
  - Analogous to taking 11 mod 5 = 1

CRC Procedure

- Select a divisor polynomial C(x), degree k
  - C(x) should be irreducible – not expressible as a product of two lower-degree polynomials
- Add k bits to message
  - Let $n(x) = m(x)x^k$ (add k 0's to m)
  - Compute r(x) = n(x) mod C(x)
  - Compute n(x) = n(x) – r(x) (will be divisible by C(x))
    - Subtraction is XOR, just set k lowest bits to r(x)!
- Checking CRC is easy
  - Do the above including the check bits at the end, make sure remainder is now 0

Mathematical steps

Actual steps in hardware/software (simple)

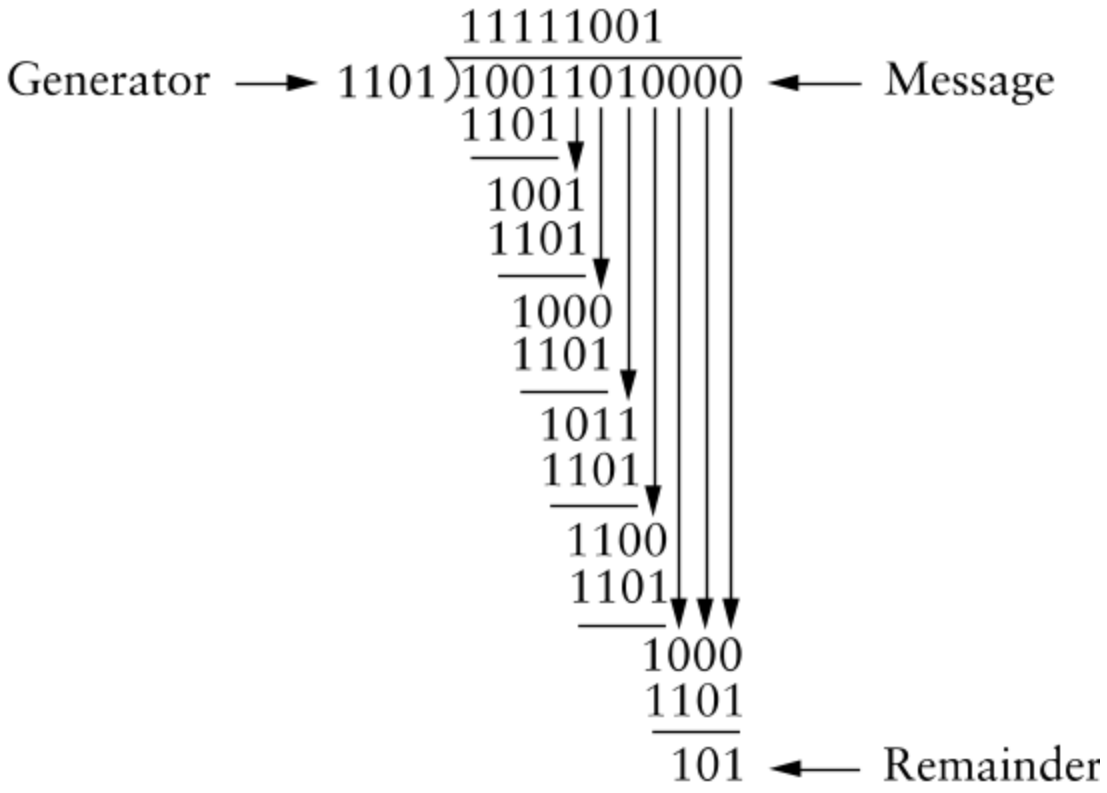
n = m << k

r = binary_long_division()
(see next slide)

Append r to message

Example – Polynomial Division

- Division where addition & subtraction is XOR

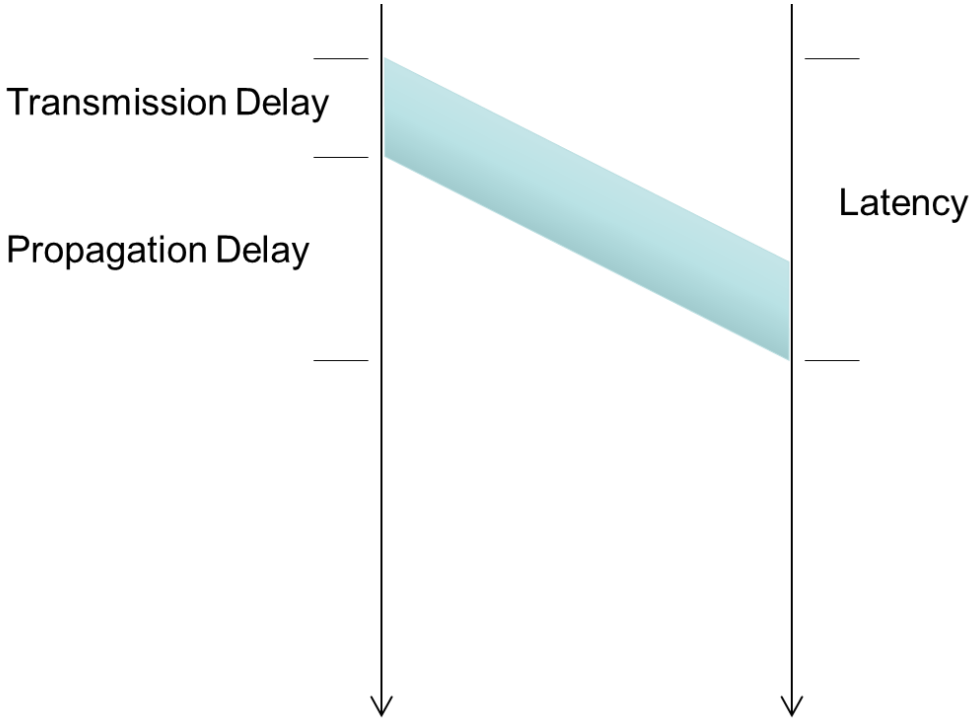


Why This Works Well

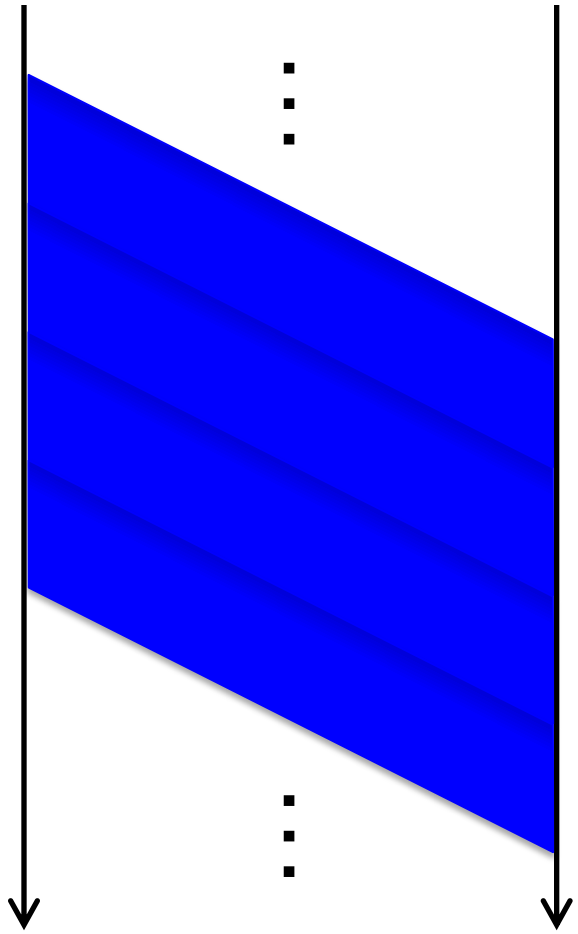
- Suppose you send m(x), recipient gets m′(x)
  - E(x) = m′(x) – m(x) (all the incorrect bits)
  - If CRC passes, C(x) divides m′(x)
  - Therefore, C(x) must divide E(x)
- Choose C(x) that doesn't divide any common errors!
  - All single-bit errors caught if x^k, x^0 coefficients in C(x) are 1
  - All 2-bit errors caught if at least 3 terms in C(x)
  - Any odd number of errors if last two terms (x + 1)
  - Any error burst less than length k caught

Data Link Layer Protocols

- Now that we understand error handling and framing…
  - What kinds of protocols for flow control and message delivery?
- First, we need to understand a bit about link performance



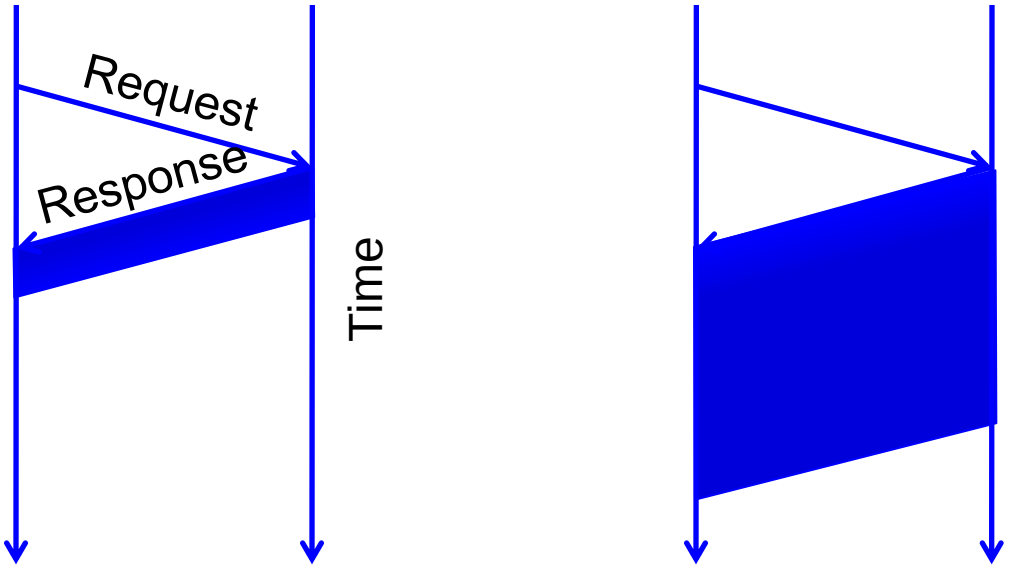
Sending Frames (Bandwidth)



Throughput: bits / s

Latency vs. Bandwidth

- How much data can we send during one Round-Trip Time (RTT)?
- *E.g.*, send request, receive file

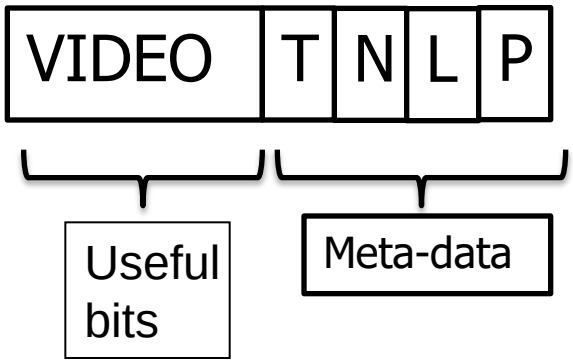


- **For small transfers, latency more important, for bulk, throughput more important**

^ You will see this over and over in computing forever! Note it now!

Performance Metrics

- Throughput - Number of bits received/unit of time
  - *e.g.* 10Mbps
- Goodput - *Useful* bits received per unit of time



- Latency – How long for message to cross network
  - Process + Queue + Transmit + Propagation
- Jitter – Variation in latency

Latency

- Processing
  - Per message, small, limits throughput
  - *e.g.* $\frac{100Mb}{s} \cdot \frac{pkt}{1500B} \cdot \frac{B}{8b} \approx 8,333 pkt/s$ or 120μs/pkt
- Queue
  - Highly variable, offered load vs outgoing b/w
- Transmission
  - Size/Bandwidth
- Propagation
  - Distance/Speed of Light

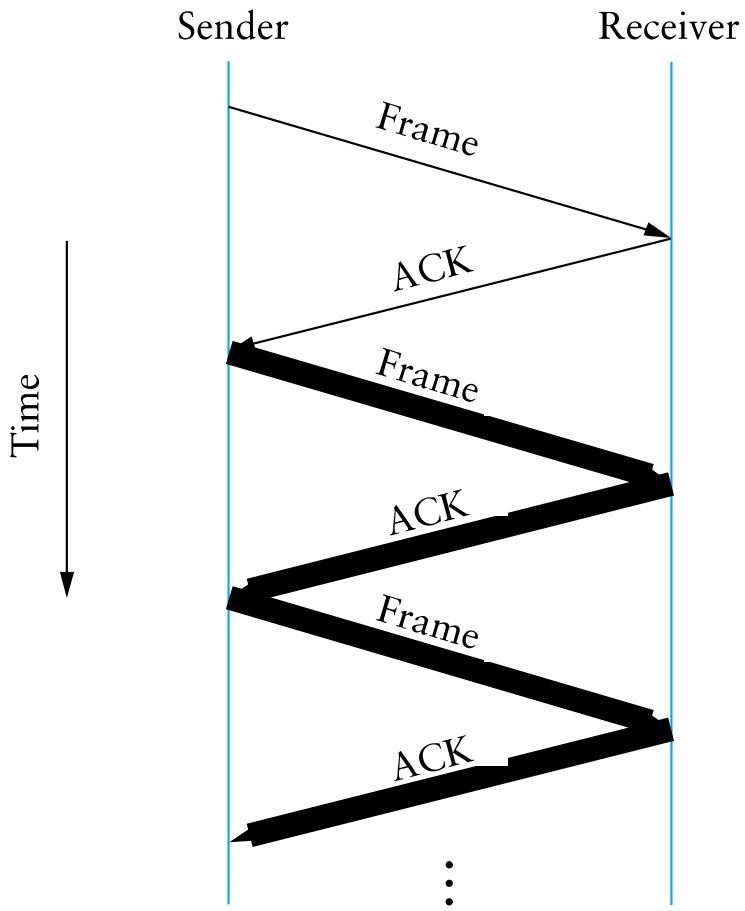
Reliable Frame Delivery

- Several sources of errors in transmission
- Error detection can discard bad frames
- Problem: if bad packets are lost, how can we ensure reliable delivery?
  - Exactly-once semantics = at least once + at most once

At Least Once Semantics

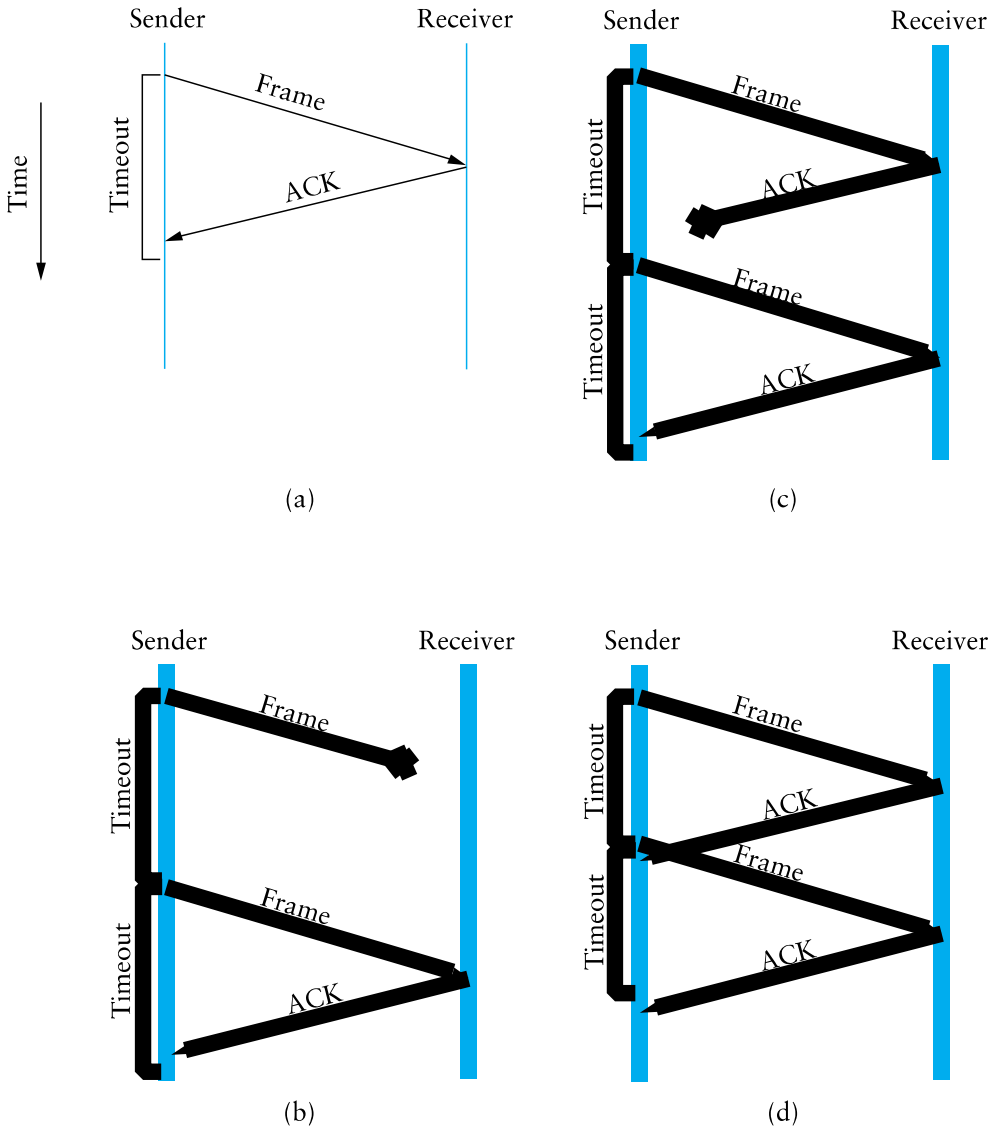
- How can the sender know packet arrived *at least once*?
  - Acknowledgments + Timeout
- Stop and Wait Protocol
  - S: Send packet, wait
  - R: Receive packet, send ACK
  - S: Receive ACK, send next packet
  - S: No ACK, timeout and retransmit

Stop-and-Wait Protocol



Ideal flow

Some Problem Scenarios

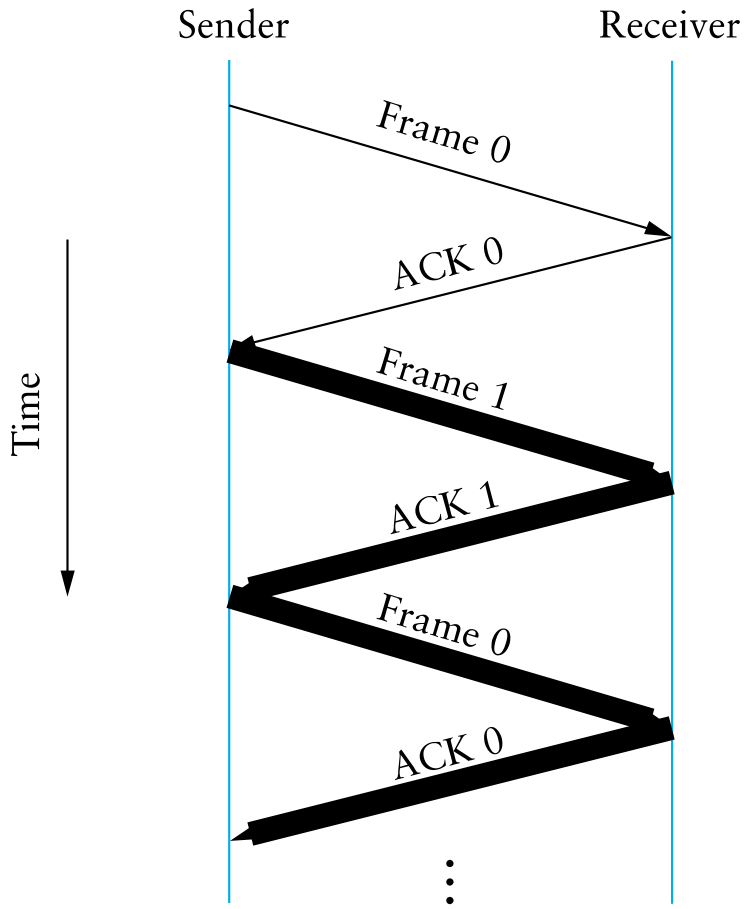


In (c) and (d), does the receiver know whether the second frame is a new frame or a re-sent first frame?

Drawbacks of Stop-and-Wait

- Duplicate data
- Duplicate acks
- Slow (channel idle most of the time!)
- May be difficult to set the timeout value

Add Sequence Numbers

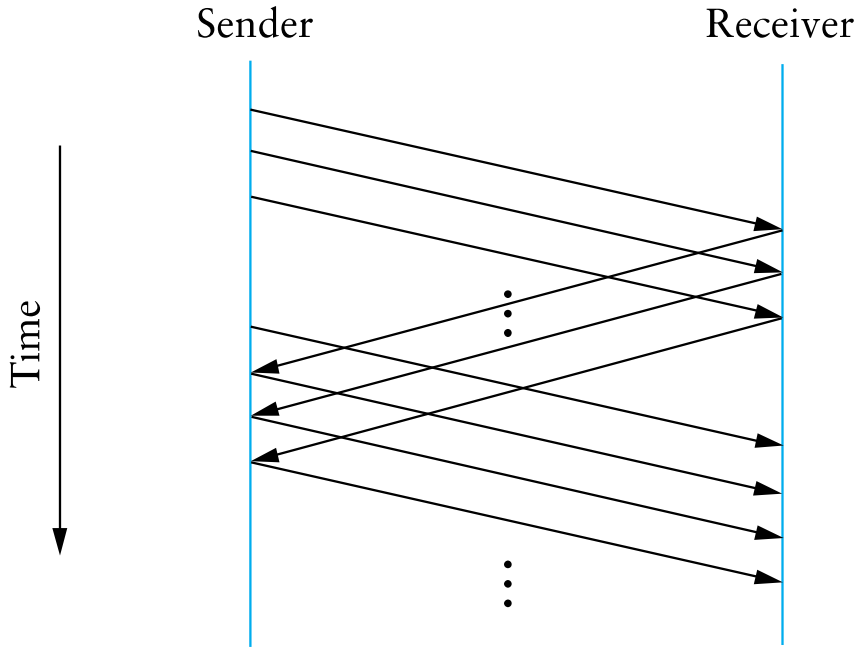


At Most Once Semantics

- How to avoid duplicates?
  - Uniquely identify each packet
  - Have receiver and sender remember
- Stop and Wait: add 1 bit to the header
  - Why is it enough?

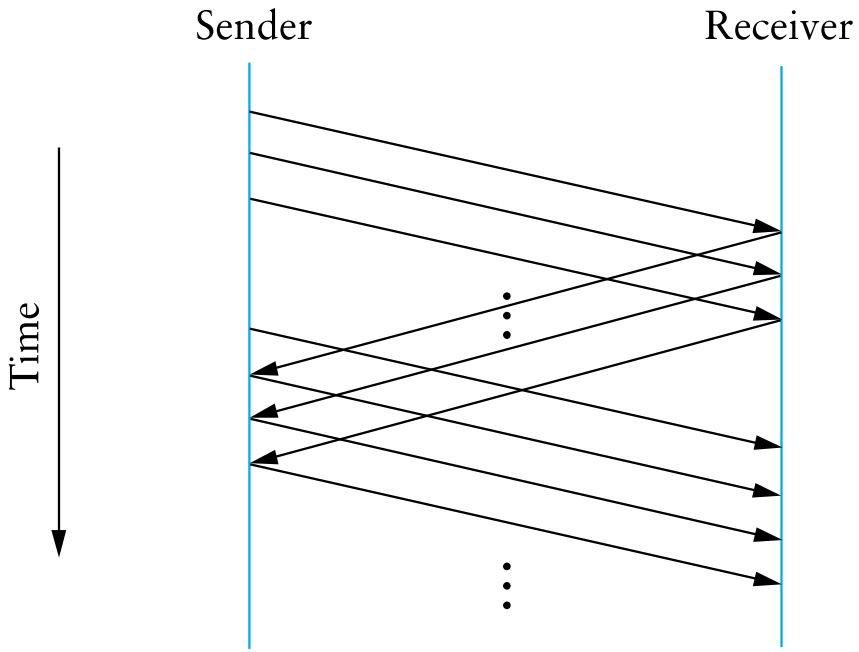
Sliding Window Protocol

- Still have the problem of keeping pipe full
  - Generalize approach with > 1-bit counter
  - Allow multiple outstanding (unACKed) frames
  - Upper bound on unACKed frames, called *window*

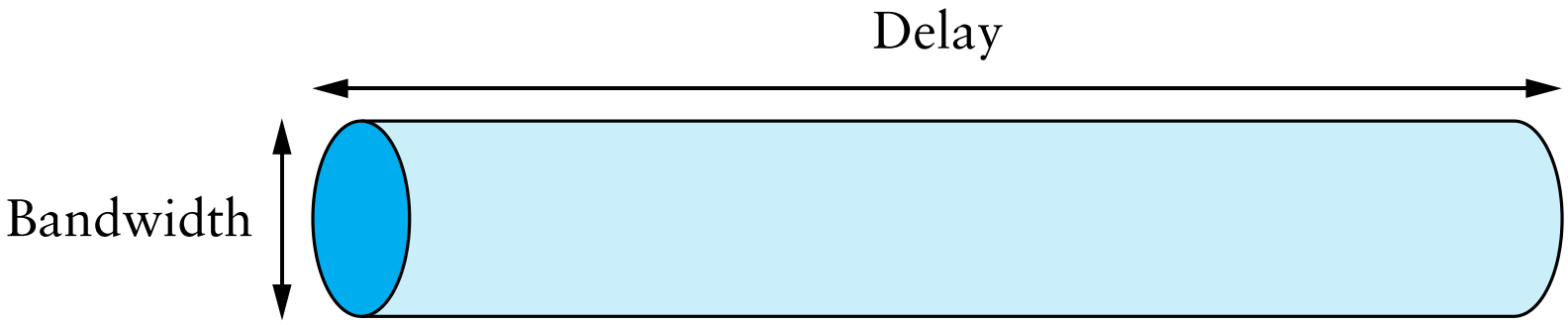


Sizing the Window

- How many bytes can we transmit in one RTT?
  - BW B/s x RTT s => "Bandwidth-Delay Product"



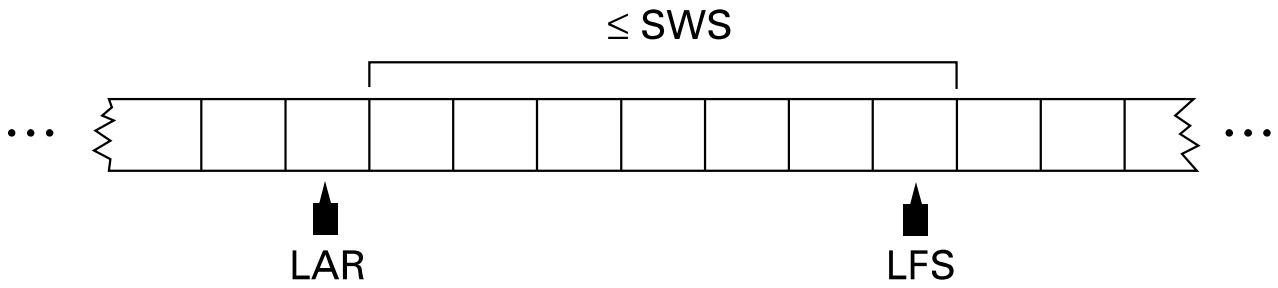
Maximizing Throughput



- Can view network as a pipe
  - For full utilization want bytes in flight ≥ bandwidth × delay
  - But don't want to overload the network
- Blast packets into the Network
- What if protocol doesn't involve bulk transfer?
  - Get throughput through concurrency – service multiple clients simultaneously

Sliding Window Sender

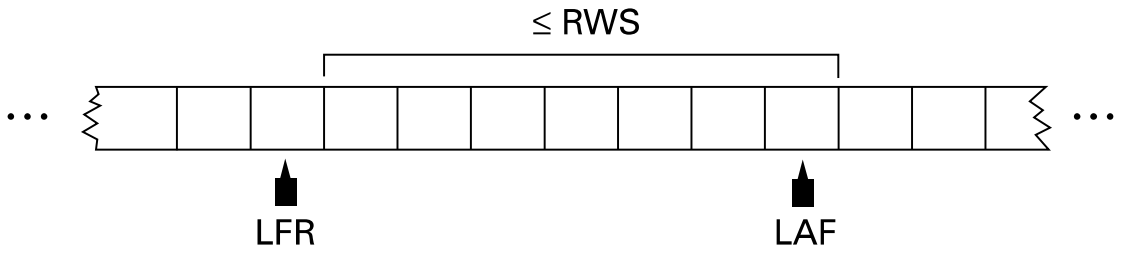
- Assign sequence number (SeqNum) to each frame
- Maintain three state variables
  - send window size (SWS)
  - last acknowledgment received (LAR)
  - last frame sent (LFS)



- Maintain invariant: LFS – LAR ≤ SWS
- Advance LAR when ACK arrives
- Buffer up to SWS frames

Sliding Window Receiver

- Maintain three state variables:
  - receive window size (RWS)
  - largest acceptable frame (LAF)
  - last frame received (LFR)



- Maintain invariant: LAF – LFR ≤ RWS
- Frame SeqNum arrives:
  - if LFR < SeqNum ≤ LAF, accept
  - if SeqNum ≤ LFR or SeqNum > LAF, discard
- Send *cumulative* ACKs

Q: Why discard packets outside of the receive window?
A: What if you get packet 1 and packet 1,000,000,000? Should you allocate a 1GB buffer and wait for the other 999,999,998 packets you think you're missing?

Tuning the Sending Window

- How big should SWS be?
  - "Fill the pipe"
- How big should RWS be?
  - $1 \leq RWS \leq SWS$
- How many distinct sequence numbers needed?
  - SWS can't be more than half of the space of valid seq#s.

Example

- We have 3-bit sequence numbers: 0, 1, 2, …, 7
- What if SWS = RWS = 7
- Sender sends 0,1,2,3,4,5,6
- All received & acked
  - Receiver advances receive window to 7,0,1,…5
  - But all acks are lost
- Sender times out and sends 0 again
- Receiver thinks it is a *new* frame with seq # 0
  - Buffers it and awaits forwarding up to network layer
- Sender later sends frame 7; receiver gets it correctly
  - Now frame 7 and stale frame 0 are passed to network layer

Analogy: Think about waiting for your order at a fast food restaurant. If you have two-digit order numbers but >100 people are waiting for food, how do you know that the "order 52" they called is you versus someone else?

Summary

- Want exactly once
  - At least once: acks + timeouts + retransmissions
  - At most once: sequence numbers
- Want efficiency
  - Sliding window