Today's Objectives

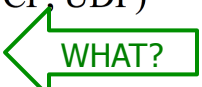
- Networking: TCP
- Threads and Synchronization

Review

- What is the OSI model?
- What are the theoretical layers?
  - What are their real-world equivalents? What do they do?
- How are packets routed?
- Online book: Introduction to Computer Networks
  - On course web site

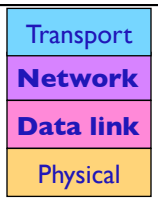
Why this whole layer 2/3 thing?

- Myth: bridges/switches simpler devices, designed before routers
- OSI Layers
  - 1: physical
  - 2: data link (nbr-nbr, e.g., Ethernet)
  - 3: network (create entire path, e.g., IP)
  - 4 end-to-end (e.g., TCP, UDP)
  - 5 and above: boring



TRANSPORT LAYER PROTOCOLS (LAYER 4)

End-to-End Protocols



- **Layers 2 & 3** (Ethernet/WiFi/IP)
  - Focus on delivering packets/frames of data to arbitrary ***hosts*** connected to the Internet
  - Routing protocols for getting packets to destination (Link State Protocol)
  - IP is best-effort delivery (no reliability)
- **Layer 4**
  - Focuses on arbitrary ***processes*** communicating together
  - Provide illusion that all processes are on one large computer
  - Can deliver data *reliably* to any process running on any host

Option 1: UDP

- User Datagram Protocol (UDP) - invented in 1980
  - Simple transport layer protocol
    - No guarantees about reliability, in-order delivery
  - "Thin veneer" on top of IP
    - Adds src/dest port numbers
    - 16-bit port number allows for identification of 65536 unique communication endpoints per host
    - Note that a single process can utilize multiple ports
    - IP addr + port number uniquely identifies all Internet endpoints
  - UDP Datagram:

Option 2: TCP

- Transmission Control Protocol (TCP) - 1974/1981
  - Reliable in-order delivery of byte streams
  - Full duplex (endpoints simultaneously send/receive)
    - Two-way traffic is permitted
    - Ex: single socket for web browser talking to web server
  - Provides flow-control
    - Ensures that sender does not overrun receiver
    - Fast server talking to slow client
  - Provides congestion control
    - Keep the sender from overrunning the network
    - Many simultaneous connections across routers (cross traffic)

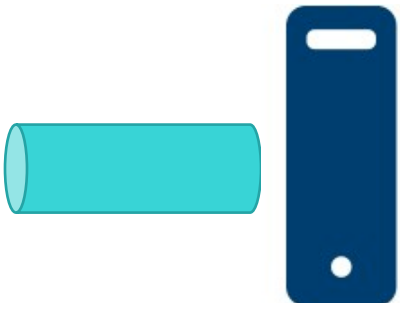
TCP Flow & Congestion Control

- Sender must determine maximum amount of data in transit that will not overrun *either* receiver or network
- Solutions?

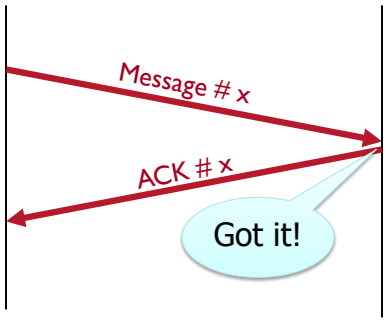
Data?

Data?

Data?



Sending Messages



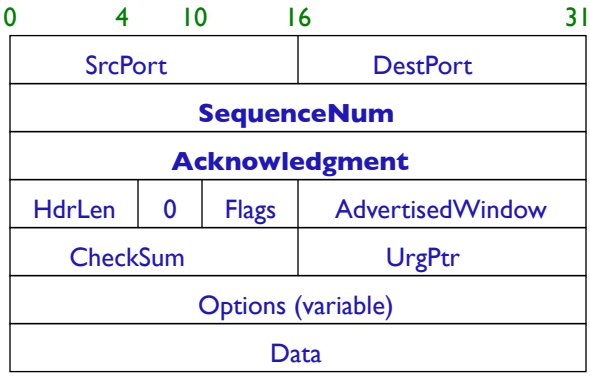
TCP Flow Control

- Sender must determine maximum amount of data in transit that will not overrun *either* receiver or network (flow and congestion control)
- Solutions for flow control:
  - Maintain "sliding window" to track data in transit
  - Size of window determined by minimum of "flow window" and "congestion window"
  - Receiver ACKs "slide" left side of window forward
    - Opens up another "slot" at right side of window for transmission

TCP "Sliding Window" Protocol Issues

- Need for connection establishment
  - No dedicated cable
- Varying round trip times over life of connection
  - Different paths, different levels of congestion
- Must be ready for very old packets to arrive
- Delay-bandwidth product highly variable
  - Amount of available buffer space at receivers also varies
- Sender has no idea what links will be traversed to receiver in advance
  - Must dynamically estimate changing end-to-end characteristics

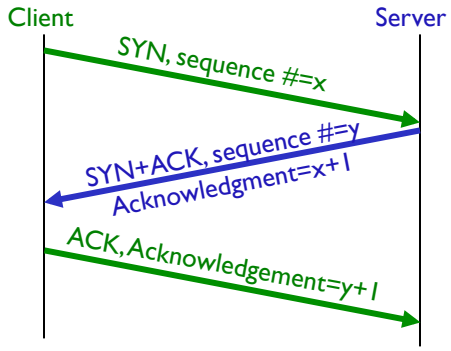
TCP Header Format



- Without options, TCP header 20 bytes
- IP header is also 20 bytes
  - Thus, typical Internet packet min of 40 bytes (+link header)

Establishing a TCP Connection

- Exchange necessary information to begin communication
- Three-way handshake
  - E.g., server listening on socket

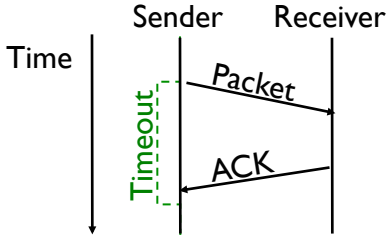


TCP Connection Teardown

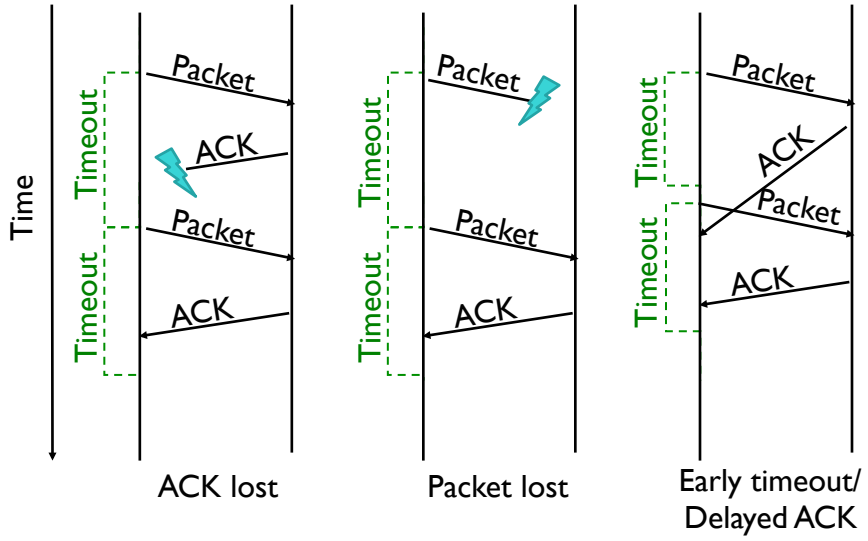
- Each side of a TCP connection can independently close the connection
  - Possible to have a half duplex connection
  - Possible problems?
  - Solutions?
- Closing process sends a `FIN` message
  - Waits for `ACK` of `FIN` to come back
  - This side of the connection is now closed

Reliability, First Cut: Stop and Wait

- Reliability, two principal mechanisms:
  - ACKs and timeouts
- Send a packet, stop and wait until acknowledgement arrives before sending next packet
- Problems?



Recovering From Error



Problems with Stop and Wait

- How to recognize a duplicate transmission?
  - Solution: Put sequence number in packet
- Performance
  - Unless Latency-Bandwidth product is very small, sender cannot "fill the pipe"
  - Solution: Sliding window protocol with dynamically changing window size

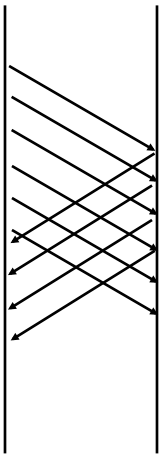
Keeping the Pipe Full

- Bandwidth-Delay product measures network capacity
  - How much data can you put into the network before the first byte reaches receiver?
- Stop and Wait: 1 data packet per RTT (round trip time)
  - Compute throughput of 1.5-Mbps link with 45-ms RTT and 1KByte data packet
  - With Stop-and-wait: 182-Kbps throughput
    - 1 Kbyte = 1024x8 bits, Throughput = 8192 bits / 45 ms = 182 Kbps

Ideally, send enough packets to fill the pipe before requiring first ACK packet

How Do We Keep the Pipe Full?

- Send multiple packets without waiting for first to be ACK'd
- Reliable, unordered delivery:
  - Send new packet after each ACK
  - Sender keeps list of unACK'd packets; resends after timeout
- Ideally, first ACK arrives immediately after pipe is filled
  - Opens up another "slot"
- Example: 10 Mbps link, 100 ms RTT:
  - How much data is needed to keep pipe full?
    - $10x10^{6}$bps * $100x10^{-3}$s = 1,000,000 bits = 125 KB



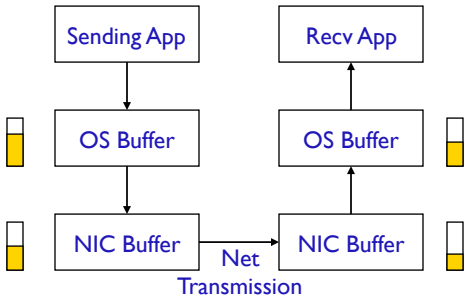
Reliable, In-Order Delivery & Flow Control

- To support in-order delivery, add sequence number
  - Receivers buffer later packets until prior packets arrive
  - When a packet arrives out of order, receiver ACKs largest sequence # received in order
    - What happens when receiver receives 1, 2, 3, 5, 6, 7?
    - Receiver ACKs 3
- Sender must still prevent buffer overflow at receiver
  - Can't forget about flow control
- Solution?
  - Sliding window with changing window size
  - Circular buffer at sender and receiver
    - # packets in transit <= buffer size
    - Advance window when sender and receiver agree packets at beginning have been successfully received

TCP Flow Control

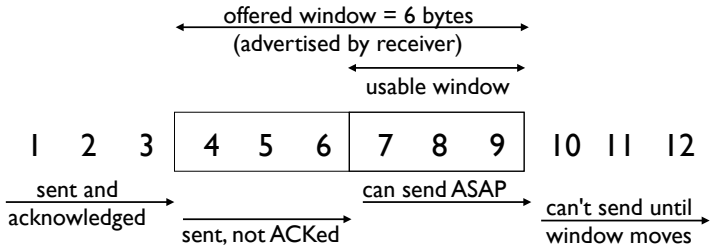
- TCP is a sliding window protocol based on byte streams, not packets
  - For window size *n*, can send up to *n* bytes without receiving an acknowledgement
  - When the data is acknowledged, window slides forward
- Each packet advertises a window size inside TCP header field
  - Number indicates number of bytes the receiver is willing to buffer

How does buffering affect flow control?



- Buffering happens at multiple points
  - Only finite space available at each location
  - System will eventually block (through backpressure)

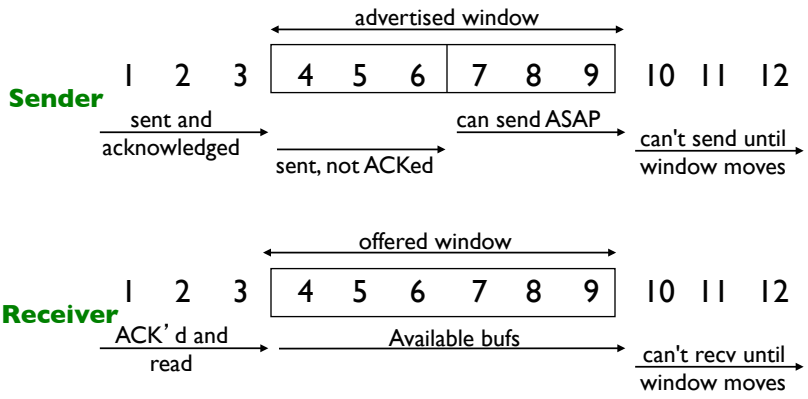
TCP Flow Control: Visualizing the Sliding Window



Left side of window advances when data is acknowledged.
Right side controlled by size of window advertisement.

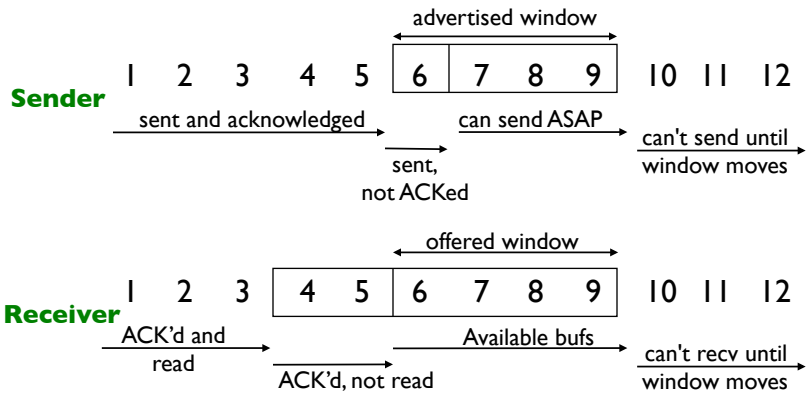
Visualizing the Window: Example

Initial State, Receiver has 6 slots to buffer data
Bytes 4, 5, 6 sent, but not yet received



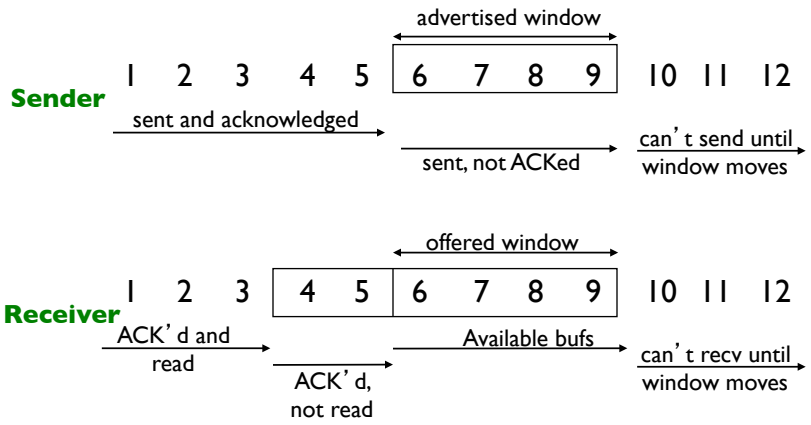
Visualizing the Window: Example

Receiver to Sender➔ ACK 5, Window 4



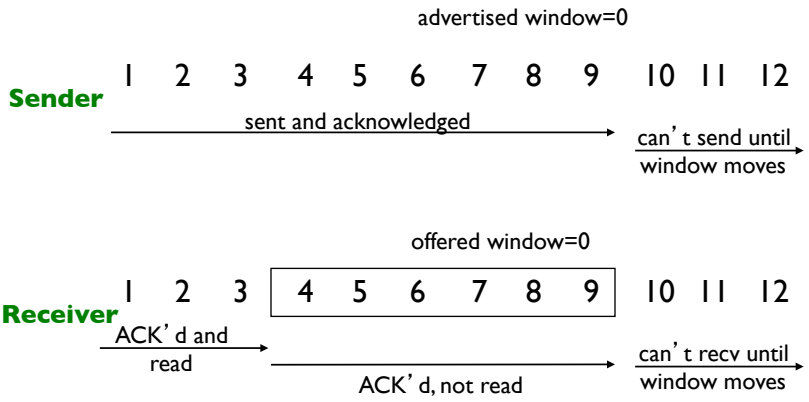
Visualizing the Window: Example

Sender to Receiver➔ Send 7, 8, 9



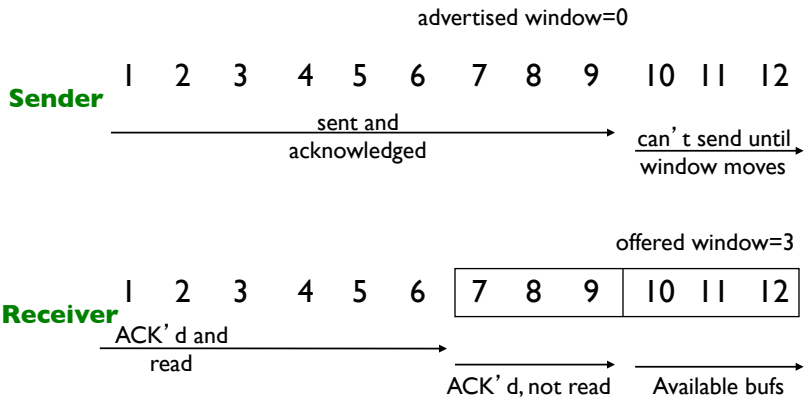
Visualizing the Window: Example

Receiver to Sender➔ ACK 9, Window 0



Visualizing the Window: Example

Receiver App reads packets 4, 5, 6
But sender has no way of knowing that more room is available!



Options for Sender Discovery of Increased Advertised Window

- Receiver sends duplicate ACK with a larger advertised window
  - Complicates receiver design
  - TCP design philosophy: keep receiver simple
- Sender periodically transmits a 1-byte packet
  - If no space available at receiver → packet dropped, no ACK
  - If additional space became available → ACK contains new advertised window
- NOTE: Advertised window expressed in *bytes* not packets!

Looking Ahead

- Read Lessons from Giant-Scale Internet Services
  - Annotate on Perusall
- Web Server Project!