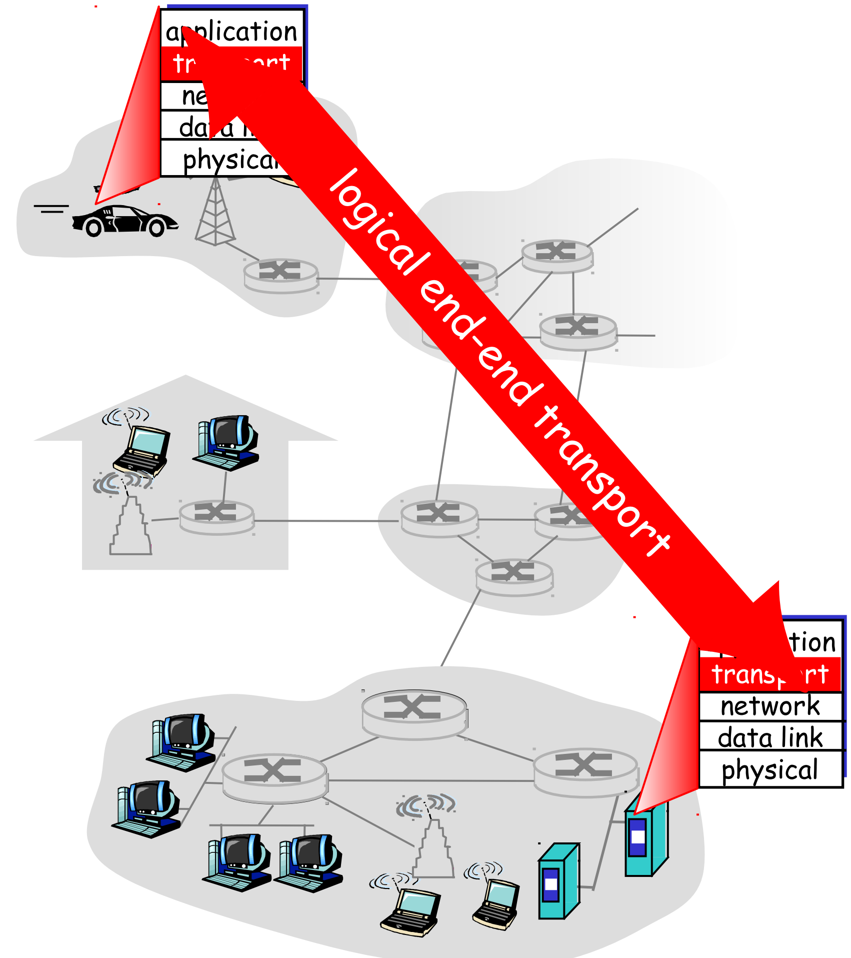
Transport protocols

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport Layer Functions

- Demux to upper layer
  - Delivering data to correct application process
- Connection setup
  - Providing a connection abstraction over a connectionless substrate
- Delivery semantics
  - Reliable or unreliable
  - Ordered or unordered
  - Unicast, multicast, anycast
- Security
- Flow control
  - Prevent overflow of receiver buffers
- Congestion control
  - Prevent overflow of network buffers
  - Avoid packet loss and packet delay

UDP: User Datagram Protocol [RFC 768]

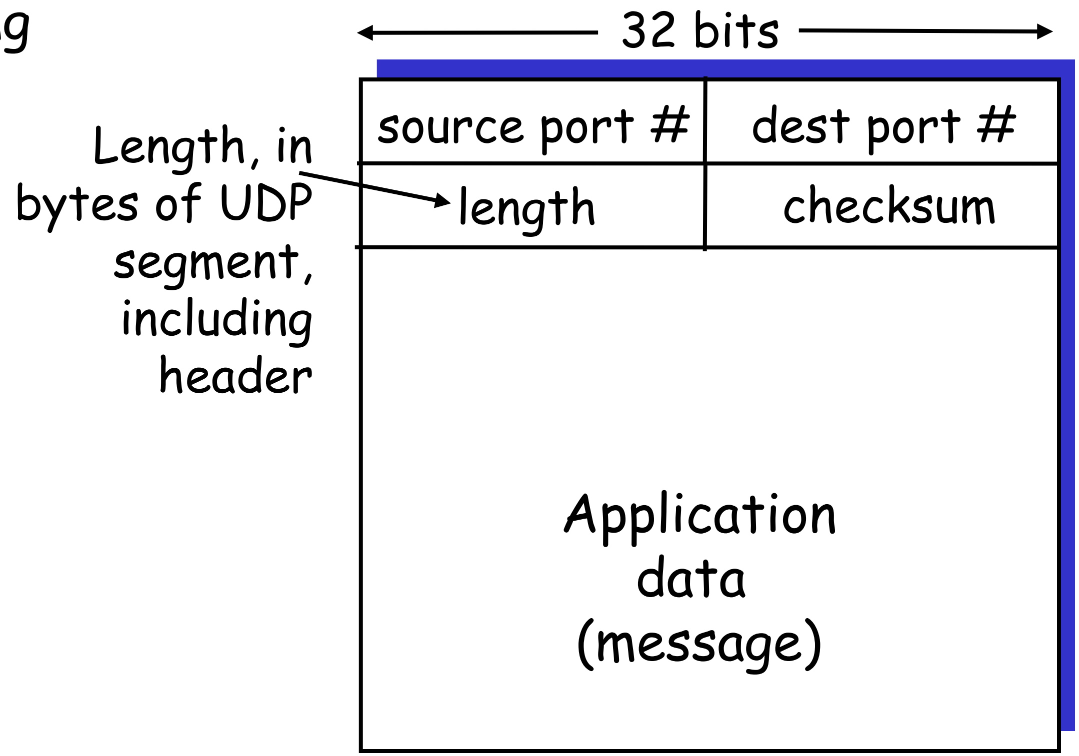
- Bare bone transport protocol
- Connectionless
  - No handshaking between sender and receiver
- Delivery semantics
  - Unordered, unreliable
  - Unicast mostly (multicast no longer supported)
- No support for security, flow control or congestion control

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: minimal state at sender, receiver
- small segment header
- can send at a fixed rate (no congestion control)

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP

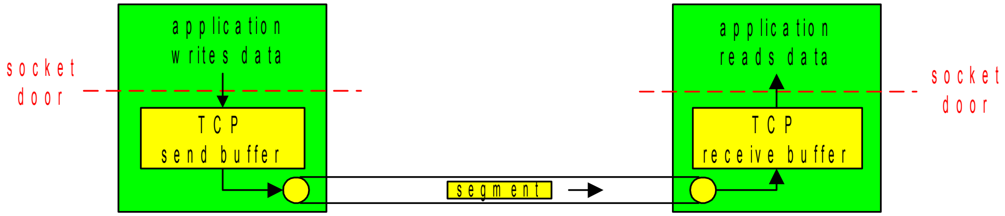


UDP segment format

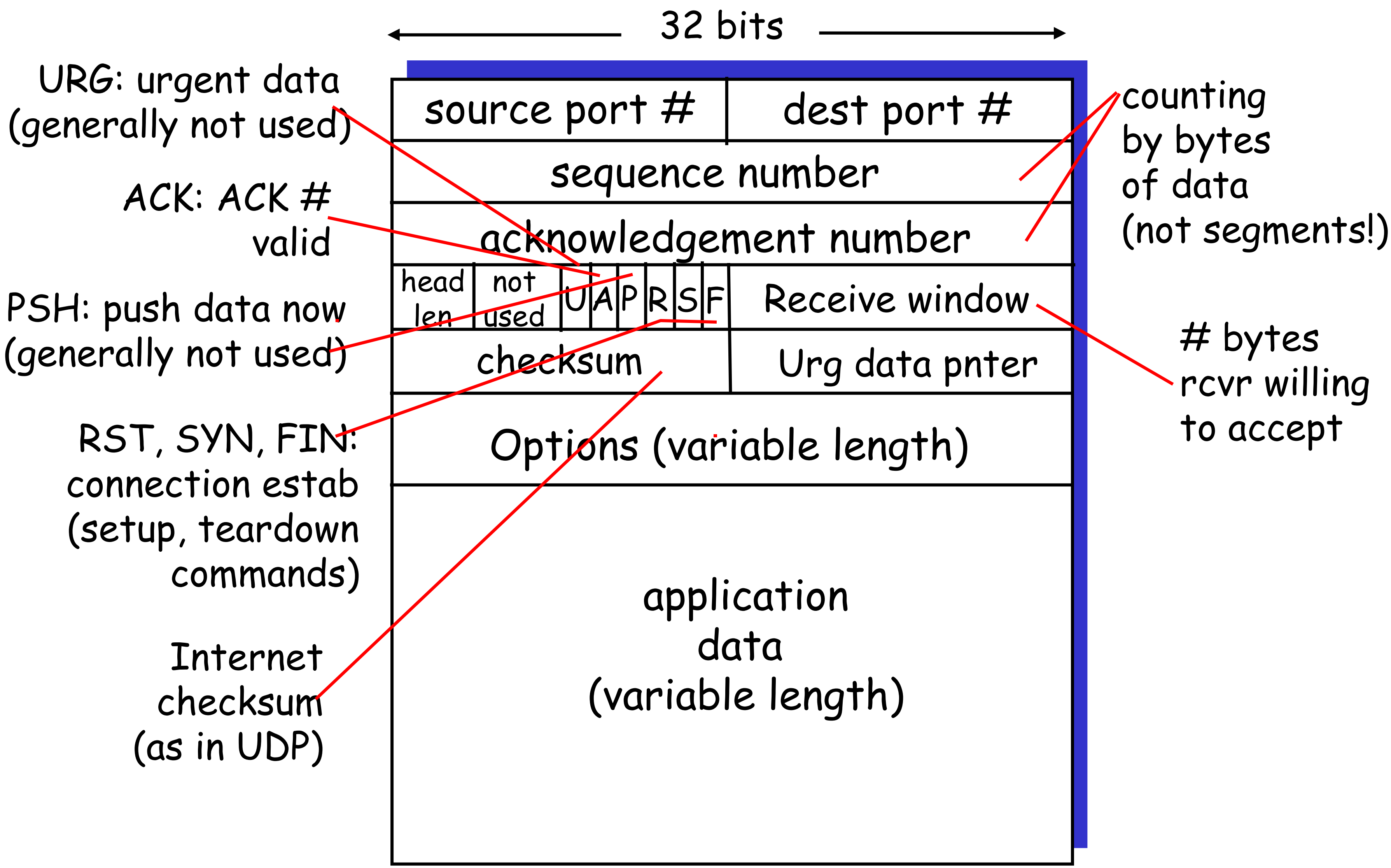
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- connection-oriented:
  - 3-way handshake to initialize sender/receiver
  - connection integrity
- reliable, in-order *byte stream:*
  - Error detection, correction
  - Retransmission
  - Duplicate detection
- full duplex:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- pipelined:
  - Support high bandwidth
- flow and congestion controlled:
  - control the size of pipe
  - sender will not overwhelm receiver or network



TCP segment structure



TCP

- TCP creates a reliable data transfer service on top of IP's unreliable service via
  - Checksum
  - Sequence numbers
  - Acknowledgments
  - Retransmissions
  - Rate limits on sender

Segment integrity via checksum

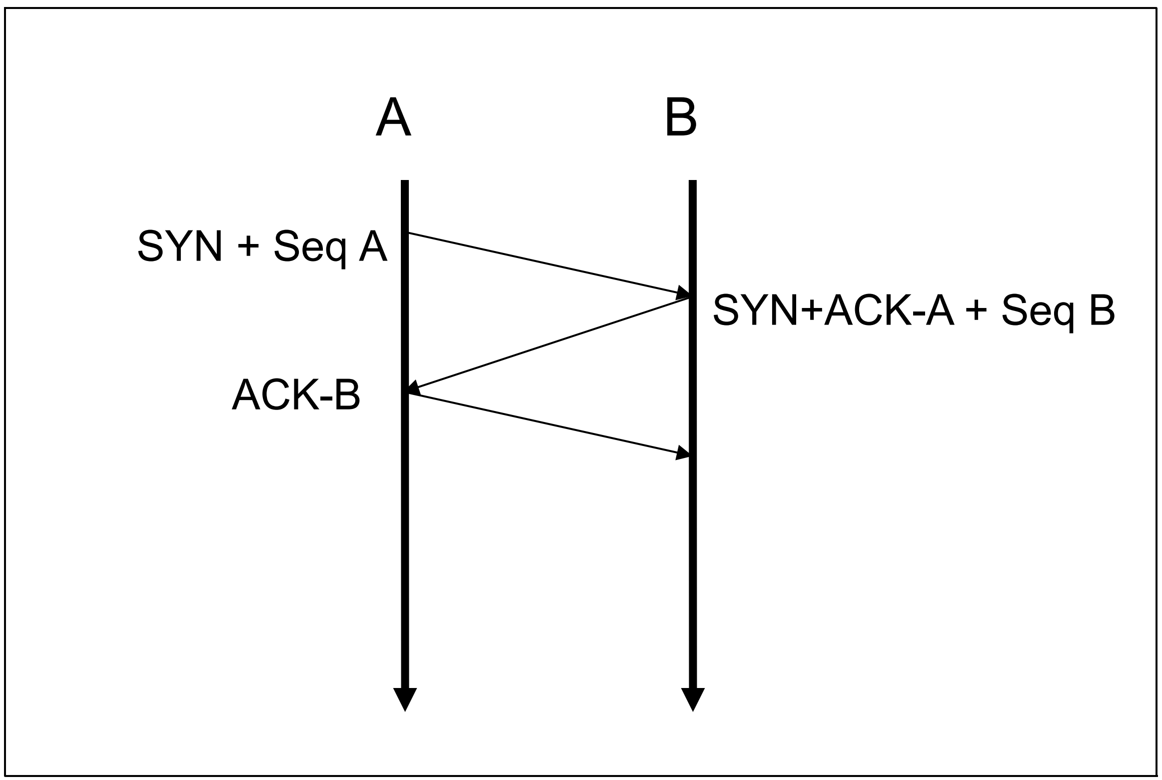
- Checksum included in header by sender
  - Generated by treating data in the packet as numbers and adding them all up
- Receiver checks checksum
  - Performs same operation as sender and checks checksum field
- Corruption detected when no match

Sequence numbers

- Data in each packet is labeled with a "unique" number
  - Establishes ordering amongst packets
  - Allows receiver to identify which packets have been received and which have not
  - Initialized during connection setup (i.e. 3-way handshake)

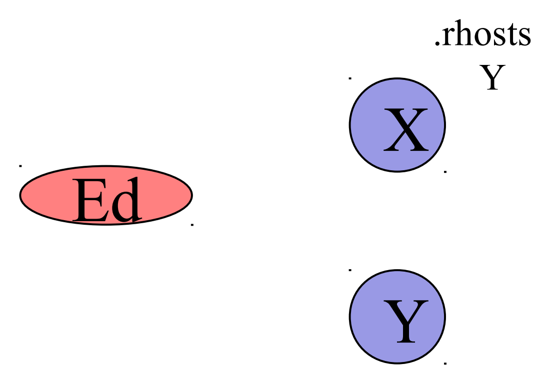
Sequence numbers

- 3-way handshake with initial sequence number selection



Sequence Numbers

- Why is selecting a random initial sequence number important?
  - Predictable ISNs allow adversary to blindly "spoof" connections from "trusted" hosts
    - Hijack TCP connections
    - Reset existing TCP connections
    - Create new connections as someone else
      - Attack popularized by K. Mitnick
      - X trusts Y
      - Logins from Y are accepted without credential check
      - Predictable ISN of X allows Evil Ed to impersonate Y and access X without credential check



Acknowledgements

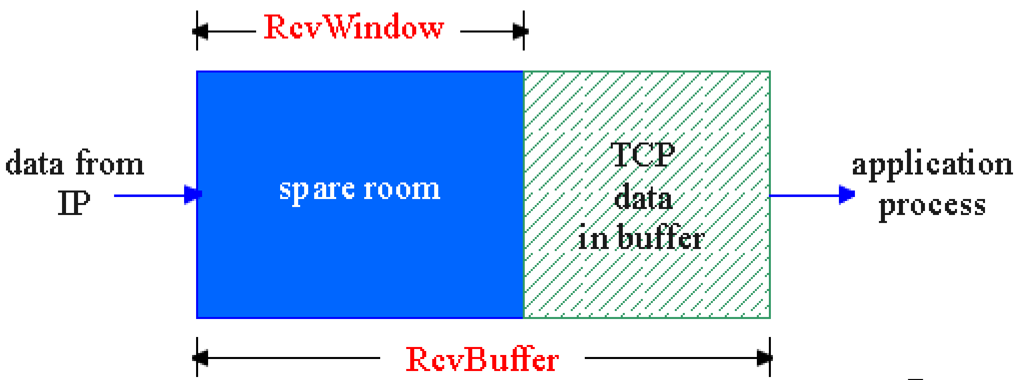
- TCP receiver sends an acknowledgement back to sender for the data it receives
  - Lets sender know to "move on"
  - Lets sender know that network has the capacity to deliver its packets

Retransmissions

- Via timeout events
  - TCP uses single retransmission timer
  - Sender sends segment and sets a timer
    - Send 1
    - Wait for Ack 2
    - No Ack 2 and timer expires
    - Send 1 again
  - Timer is based on measured round-trip times and round-trip time variations
- Via missing acknowledgements
  - If receiver reports it has received packets 1, 3, 4, and 5, sender automatically resends 2 before timeout

Rate limits on sender: Flow control

- Receiver has a finite buffer
  - App process may be slow reading it
  - Flow control to make sure sender won't overflow it
  - Match the send rate to the receiving app's drain rate
- Rcvr advertises spare room in buffer by including value of **`RcvWindow`** in each segment/ACK
  - Also known as the "advertised" window
  - Sender limits unACKed data to **`RcvWindow`** to avoid overflow

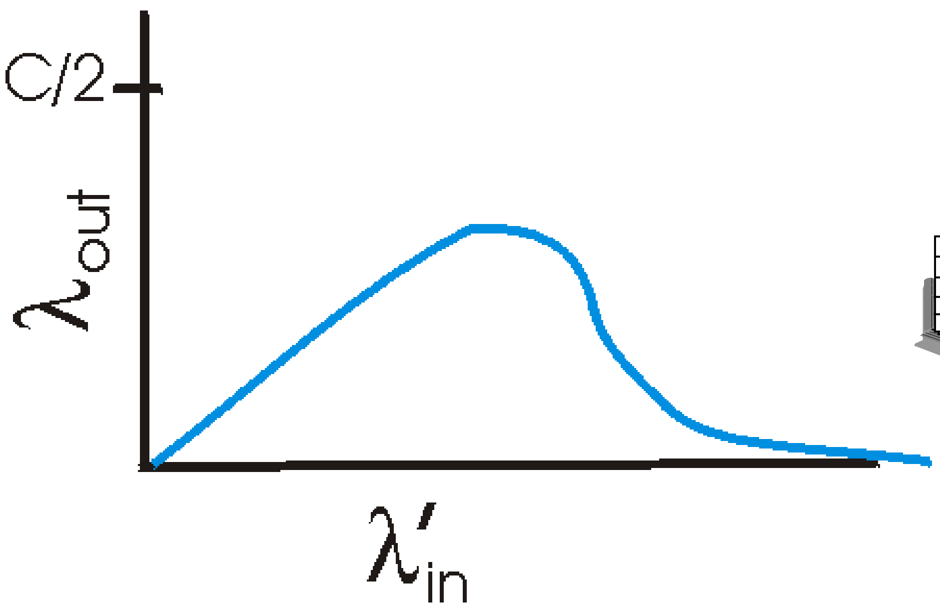


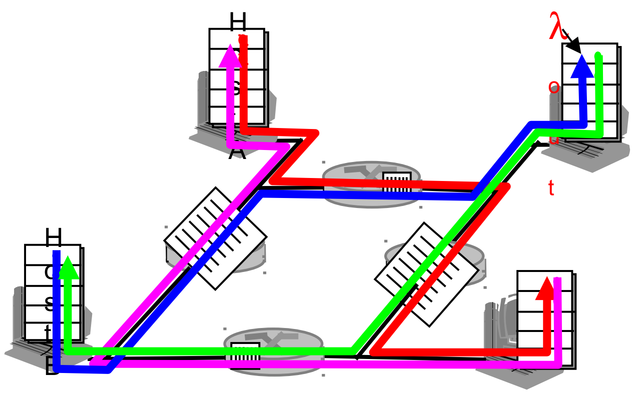
Rate limits on sender: Congestion Control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

Congestion collapse scenario





"Cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

Congestion Collapse

- Increase in network load results in decrease of useful work
  - Spurious retransmissions of packets still in flight
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Packets that are delayed on long queues

Congestion control approaches

- End-host vs. network controlled
  - End-host: Hosts trusted to adjust rate based on detected congestion
  - Network controlled: Hosts untrusted, instead have network adjust rates at congestion points
- Implicit vs. explicit network feedback
  - Implicit: infer congestion from packet loss or delay
  - Explicit: signalled from network
- Given what you know of Internet design, which one is used on the Internet?

TCP Congestion Control

- Mainly end-host, window-based congestion control
  - Only place to really prevent collapse is at end-host
  - Reduce sender window when congestion is perceived
  - Increase sender window otherwise (probe for bandwidth)

TCP congestion control basics

- Keep a congestion window, (**cwnd**)
  - Denotes how much network is able to absorb
    - "Size of the pipe"
    - Make **cwnd** as large as possible without loss
    - TCP "probes" for usable bandwidth continuously
      - *increase* **cwnd** until loss (congestion)
      - *decrease* **cwnd** upon loss ,then begin probing (increasing) again
- Recall receiver's advertised window (`rcv_wnd`)
- Sender's maximum window
  - min(rcv_wnd,cwnd)

TCP slow start

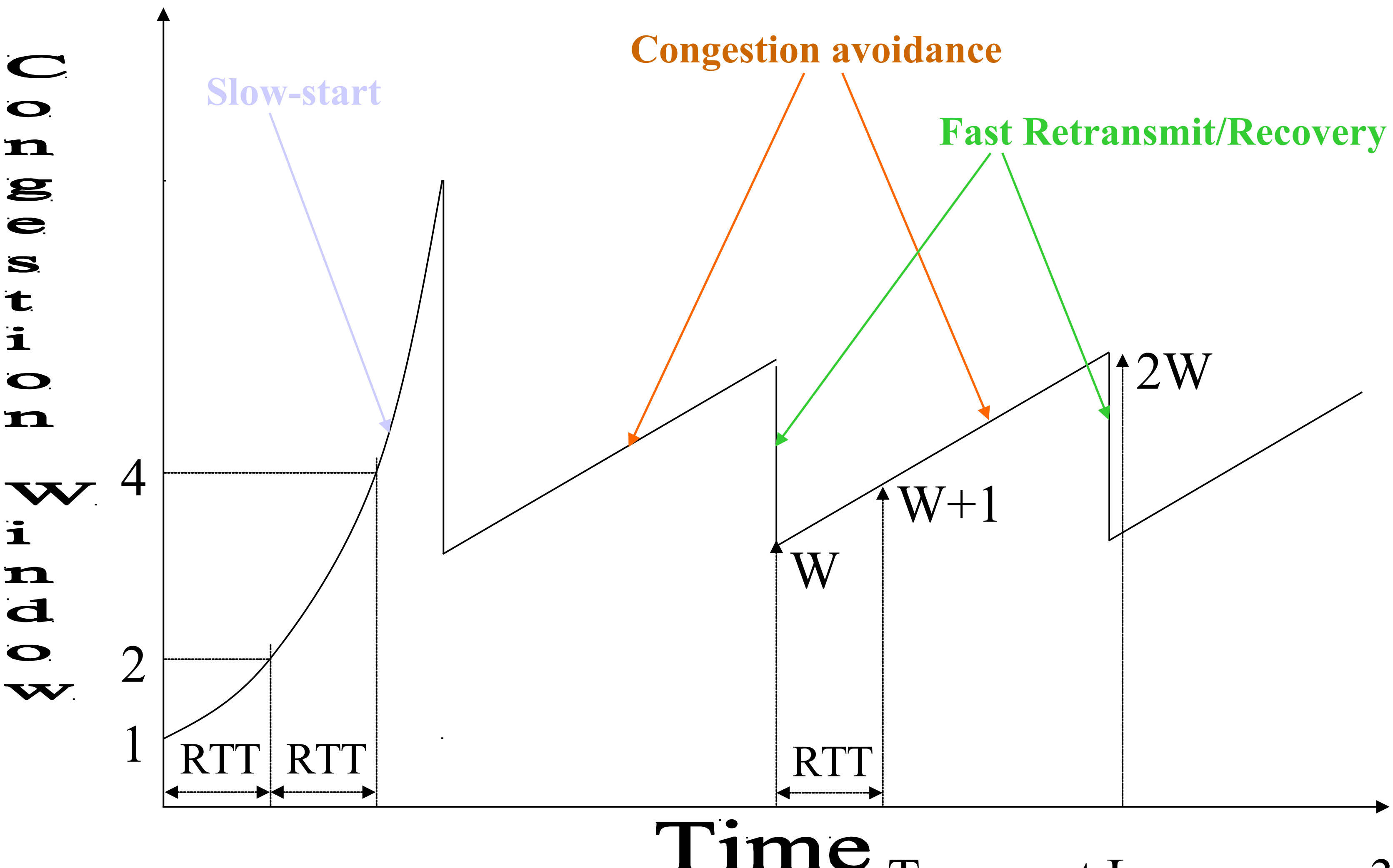
- When connection begins, increase rate exponentially fast until first loss event
  - `cwnd` = 1 for 1st RTT
  - `cwnd` = 2 for 2nd RTT
  - `cwnd` = 4 for 3rd RTT
- When connection begins, `cwnd` = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- Available bandwidth may be much larger than MSS/RTT
  - desirable to quickly ramp up to respectable rate

TCP congestion avoidance

Q: When should the exponential increase stop?

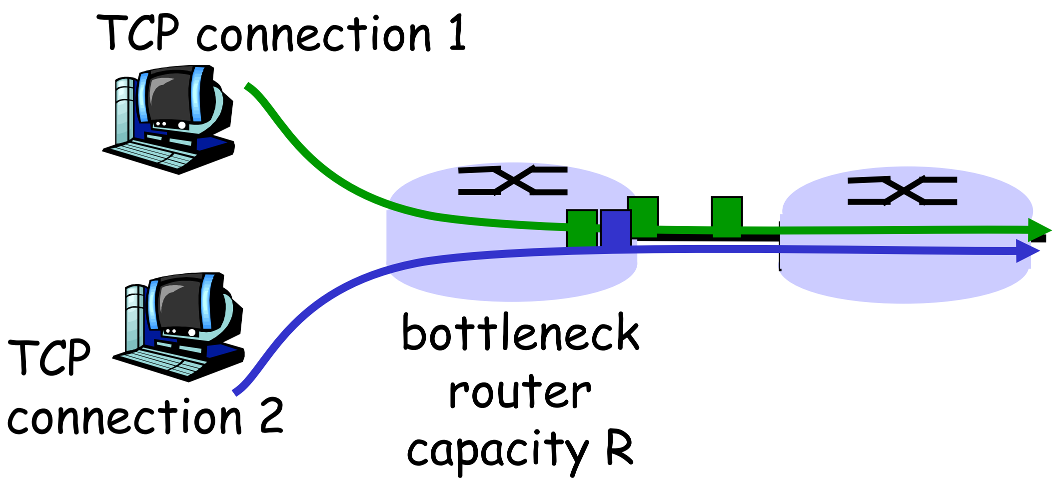
- If loss occurs when cwnd = W
  - Network can handle 0.5W ~ W segments
  - Cut cwnd in half, grow window more slowly
  - Grow cwnd by 1 every round-trip time
  - Results in additive increase

TCP throughput



Goals revisited: TCP Fairness

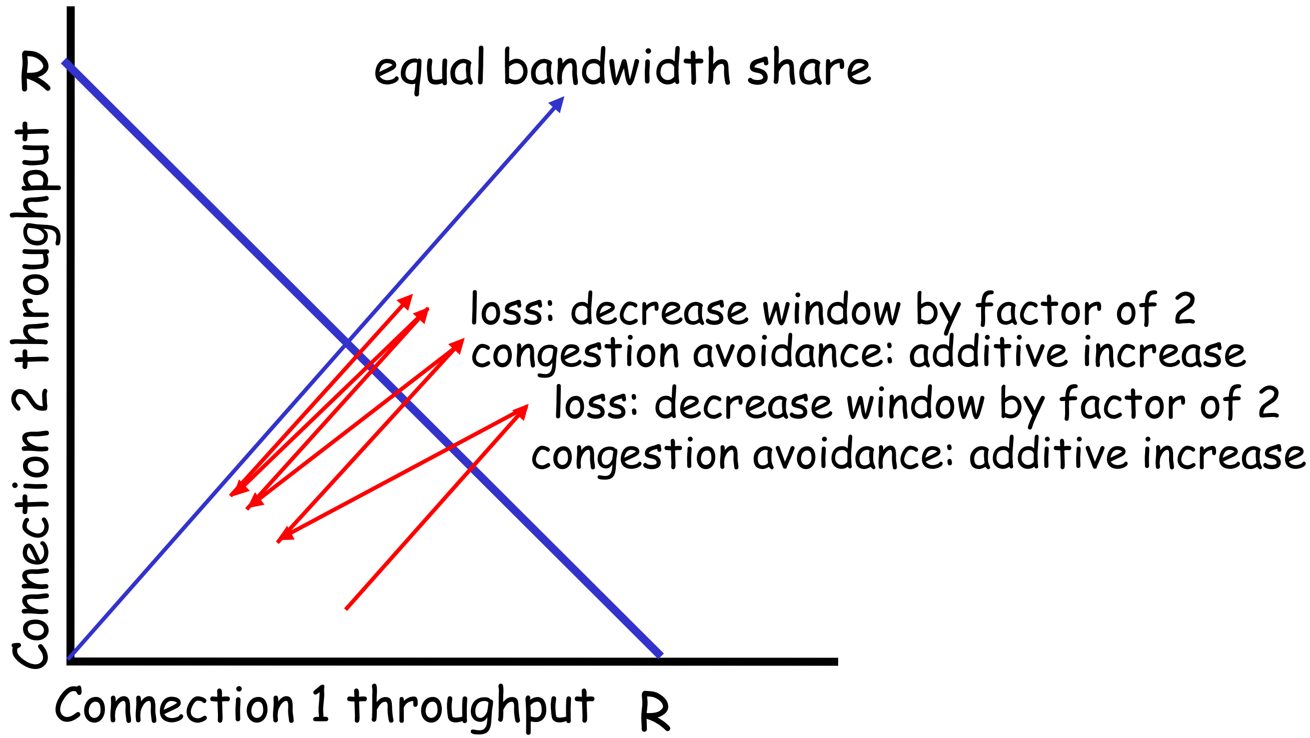
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



- Does TCP's congestion control algorithm promote fairness between flows?

Why is TCP fair?

Additive increase gives slope of 1, as throughout increases equally
Multiplicative decrease decreases throughput proportionally



Caveats to "fairness"

Fairness and UDP

- Multimedia apps often use UDP
  - pump audio/video at constant rate, tolerate packet loss
  - negatively impacts TCP flows

Fairness and parallel TCP connections

- Application opening multiple TCP connections between 2 hosts
  - Common in Web browsers
  - Link of rate R supporting 9 connections;
    - new app asks for 1 TCP, gets rate R/10
    - new app asks for 11 TCPs, gets R/2 !

Advanced transport topics

- Long fat pipes
  - Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
  - BW*Delay = 10Gbs * 0.1s = 1Gbit
    - In packets, W=83,333
    - In bytes, 1Gbit/8 = 125MB
  - Problem #1: 16-bit advertised window field (in bytes)
    - Maximum of 64KB !!
    - Solution: TCP window scaling option
      - Scaling factor on advertised window specifies # of bits to shift to the left
      - Scaling factor exchanged during connection setup

Advanced transport topics

- Long fat pipes
  - Problem #2: 32-bit sequence/ack number wraparound
    - Recall maximum window must be less than ½ seq. no. space
    - 10Mbs: 57 min., 100Mbs: 6 min., 622Mbs: 55 sec. < MSL!
    - Use timestamp option to disambiguate
    - TCP sequence number wraparound (TCP PAWS)

Advanced transport topics

- Long fat pipes
  - Problem #3: TCP Sawtooth for large W
    - For sawtooth W to 2W
    - Packets xferred in sawtooth
      - W + (W+1) + (W+2) …. + 2W = (3W/2) * (W+1) = 1.5W(W+1)
      - For W=83,333
        - » Packets xferred in sawtooth between losses = 10.4 billion
        - » Loss rate = 1 packet loss per sawtooth ➜ L = 10-10 Wow
    - Sawtooth length = W*RTT
      - For W=83,333 and RTT=100ms, sawtooth length over 2 hours
      - Average connection throughput ¾ of capacity
    - Solution: new versions of TCP for high-speed
      - HS-TCP, FAST TCP, etc.

Advanced transport topics

- Short transfers slow
  - Flows timeout on loss if cwnd < 3
    - Change dupack threshold for small cwnd
  - 3-4 packet flows (most HTTP transfers) need 2-3 round-trips to complete
    - Use larger initial cwnd (IETF approved initial cwnd = 3 or 4)

Advanced transport topics

- Security
  - Layer underneath application layer and above transport layer (See Chapter 8)
  - SSL, TLS
  - Provides TCP/IP connection the following….
    - Data encryption
    - Server authentication
    - Message integrity
    - Optional client authentication
  - Original implementation: Secure Sockets Layer (SSL)
    - Netscape (circa 1994)
    - http://www.openssl.org/ for more information
    - Submitted to W3 and IETF
  - New version: Transport Layer Security (TLS)
    - http://www.ietf.org/html.charters/tls-charter.html

Extra slides

Advanced transport topics

- Better congestion control algorithms
  - TCP increases rate until loss
  - TCP Vegas: avoid losses by backing off sending rate when delays increase
- Non-TCP traffic
  - Multimedia applications do not work well over TCP's sawtooth
  - TCP-friendly rate control (TFRC)
  - Derive smooth, stable equilibrium rate via equations based on loss rate

Advanced transport topics

- Explicit network congestion signalling
  - TCP uses implicit information to fix sender's rate
    - Packet loss reduces rate
    - Successful packet transmissions increase rate
  - ATM
    - Explicitly signal rate from network elements
  - TCP with ECN
    - Add bit in IP header to signal congestion (hybrid between TCP approach and ATM approach)
    - Actively detect and signal congestion beforehand at routers

Advanced transport topics

- Non-responsive, aggressive applications
  - Applications written to take advantage of network resources (multiple TCP connections)
  - Network-level enforcement, end-host enforcement of fairness
- Wireless networks
  - TCP infers loss on wireless links as congestion and backs off
  - Add link-layer retransmission and explicit loss notification (to squelch RTO)