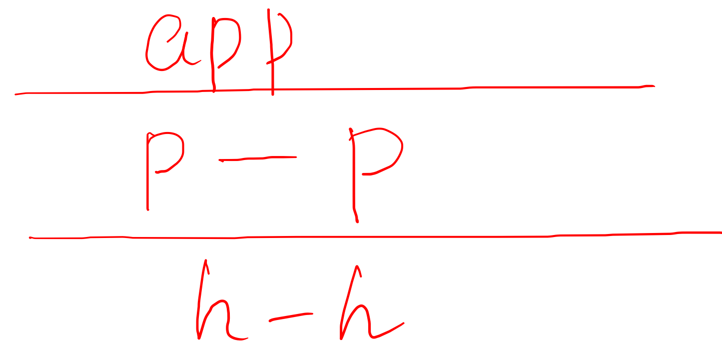
Design and Engineering of Computer Systems

Lecture 23: Transport protocols

Mythili Vutukuru

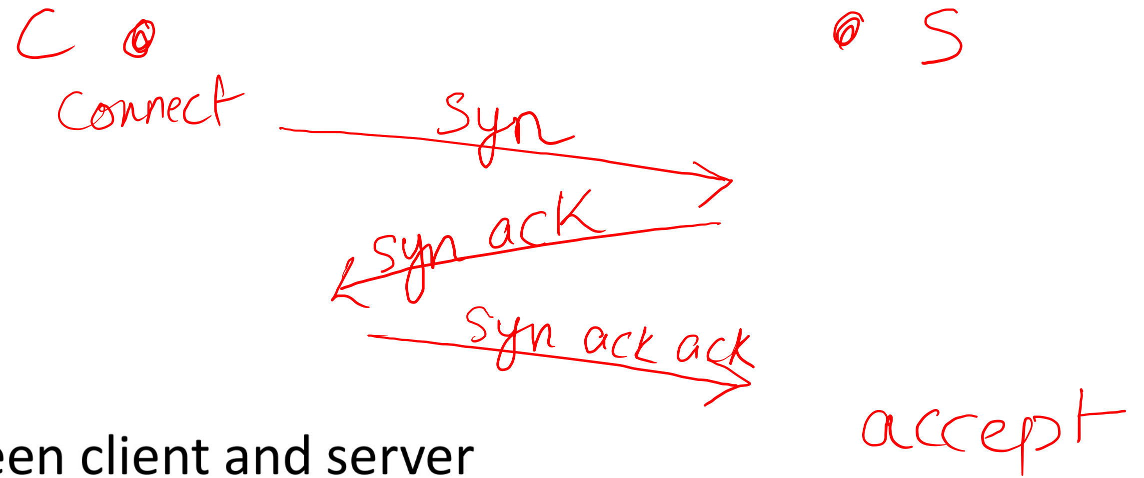
IIT Bombay

Transport layer



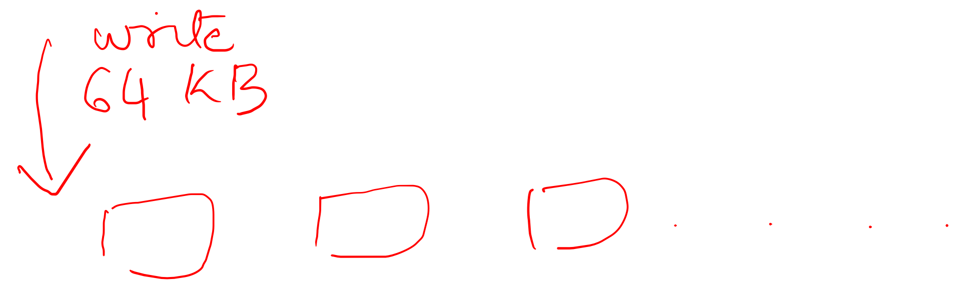
- IP layer provides host-to-host delivery of IP datagrams
  - Packets can get dropped, no reliability guarantees
- Transport layer deals with process-to-process delivery of messages
  - TCP (Transmission Control Protocol) guarantees reliable in-order delivery
  - UDP (User Datagram Protocol) does not guarantee any reliability
  - SCTP (Stream Control Transmission Protocol) provides multiple reliable streams over a connection
  - Choice depends on application requirements
- Transport protocols run only at end hosts (end to end argument)
  - Application layer writes messages into sockets, OS does transport layer processing, send packet over network
  - NIC receives packet, OS does transport layer processing, application reads message from socket

TCP connection setup

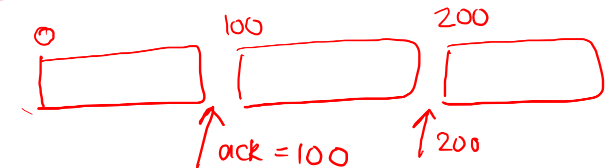


- TCP is connection-based protocol
  - End-to-end connection established between client and server
  - IP routers on path not aware of connection, only forward datagrams
- Connection establishment via 3-way TCP handshake
  - Server opens listen socket and waits to accept, client starts connect system call
  - Client’s TCP sends special SYN packet to server
  - Server replies with SYN ACK, client replies back with SYN ACK ACK
  - Connect and accept system calls return after 3-way handshake completes
- After connection established, client and server can exchange data in both directions of connection
- When data transfer done, send FIN and FIN ACK from each side to tear down connection
- UDP has no such concept of connection setup, just send packets directly

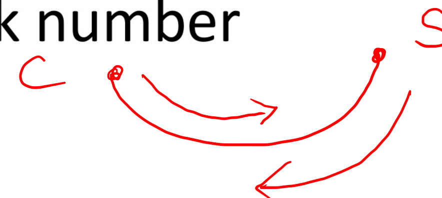
Transport layer segments



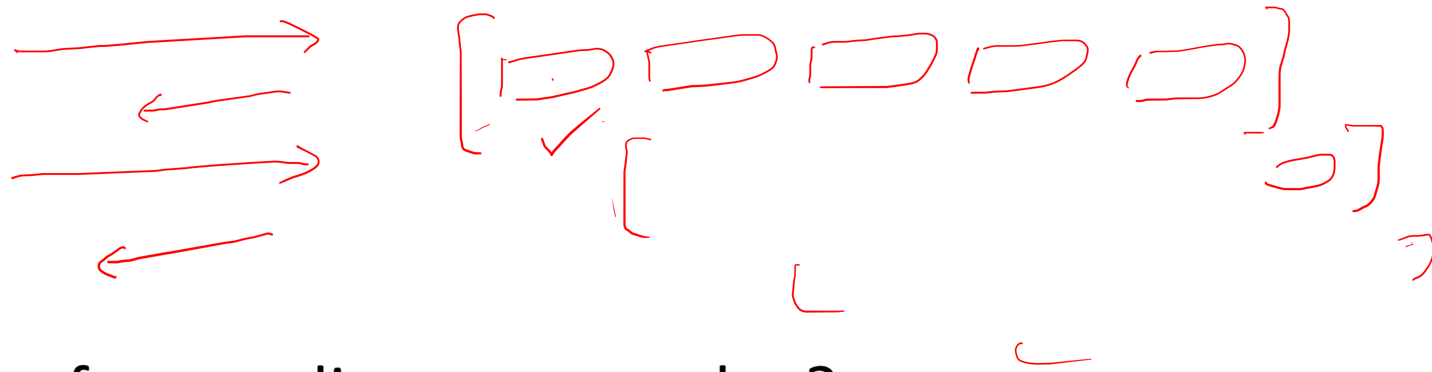
- Segmentation: message written into socket is split into chunks of MSS (maximum segment size), headers added and sent over network
  - MSS depends on underlying link technologies, which limit max packet size
  - Message boundaries may not be preserved in segments (e.g., one message may span many segments)
- TCP/UDP add source/destination port numbers to header, to help identify different sockets (source/destination IP address is part of IP layer header)
- How to ensure reliability? TCP adds sequence number and acknowledgment number in header
  - Sender puts sequence number of the starting byte present in the packet
  - Receiver replies with sequence number of the next byte it is expecting
  - Receiver's ack is cumulative, and indicates that everything up to that sequence number has been received



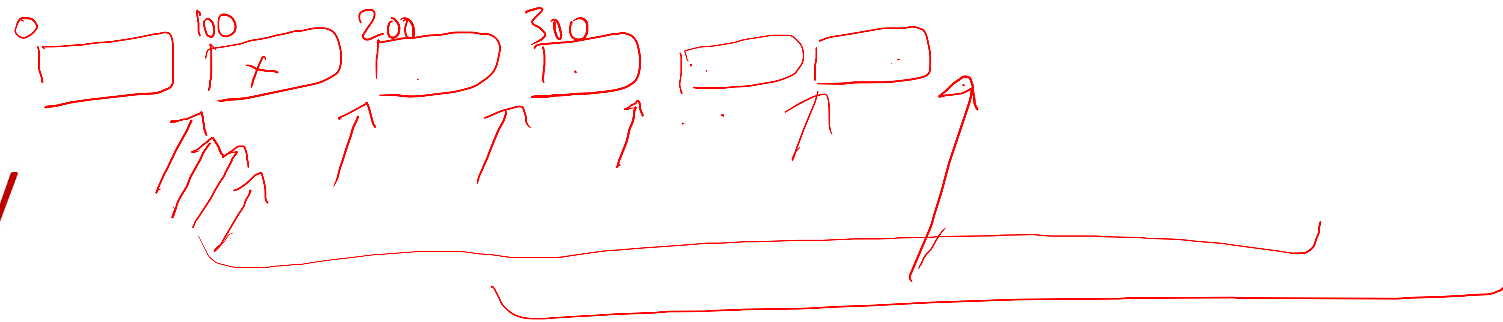
- TCP is bidirectional stream, each side sends a sequence number and ack number
  - Ack piggybacked with data in other direction, or sent in a separate packet
- Other fields in TCP/UDP segment header: packet size, check sum



Sliding window



- Should TCP sender wait for ack after sending every packet?
  - Sending packet and receiving ack takes time, sender and network idle
  - This design is called stop-and-wait, not very efficient, hence not used
- Instead, sender sends a window of W bytes before waiting for ack
- Sliding window mechanism: once ack comes for some packets, window slides forward and more packets are sent
  - Maximum number of unacknowledged bytes limited by window size W
- What value of W should be used?
  - Too large W, network/receiver can get overwhelmed, drop packets
  - Too small W, not utilizing resources properly



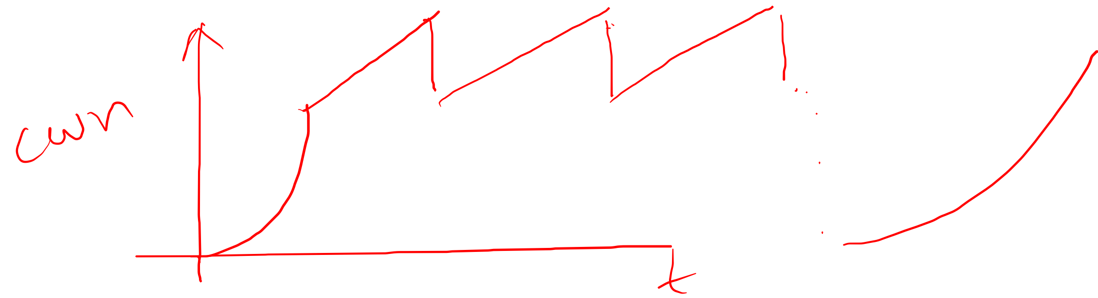
Reliability

- TCP sender transmits multiple segments, pauses if window is full
- Upon receiving a TCP segment, receiver sends ack back to sender
  - Ack sequence number is next in-order byte number expected
  - Out-of-order segments received are not reflected in ack number
- When sender receives ack, window slides forward, more data can be sent
- If a segment is lost, will result in duplicate acks from receiver (dupack)
  - A single dupack can also be due to reordering, so sender does not panic
  - If 3 dupacks for a sequence number, sender infers loss, retransmits lost segment
- What if severe congestion, all segments/acks are lost?
  - Sender maintains a retransmission timer for every segment
  - On timer expiry, timeout and retransmit everything
- What if data received at receiver, but ack is lost?
  - Sender retransmits segment unnecessarily, receiver identifies and discards duplicates
- Receiver assembles segments, sorts by sequence number, delivers to app in order

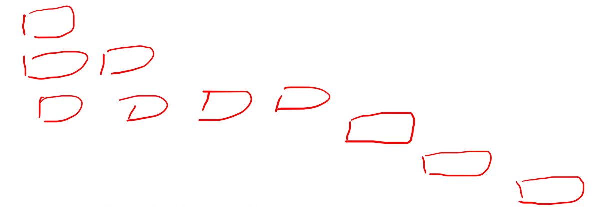
Bandwidth delay product

- How to compute window size in sliding window protocol?
- Consider the following toy example
  - Suppose network can send 10 packets/sec (bandwidth)
  - Round trip time (RTT) is 2 seconds, i.e., it takes 2 seconds for a packet to reach receiver and ack to come back
  - After sender sends 20 packets (bandwidth delay product), the ack for the first packet would have come back, and sender can send more
- Ideal sliding window size = bandwidth delay product (BDP) of connection
  - If window size > BDP, congestion in network
  - If window size < BDP, sender is idle
  - But BDP is hard to estimate (bandwidth and RTT highly variable)
- TCP sender computes congestion window size (cwnd) using heuristics

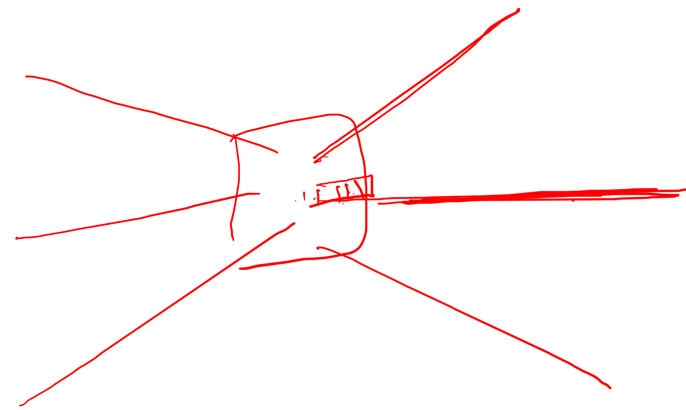
Congestion control



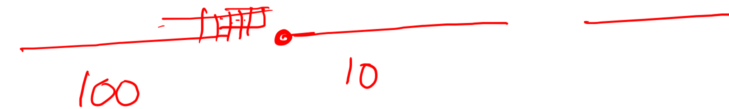
- Ideally, sender sets cwnd to be BDP, but BDP is difficult to estimate
- Instead, sender relies on feedback from network to adjust cwnd
  - If packets are going through, maybe cwnd is below BDP, send more
  - If packets are getting lost, cwnd may have crossed BDP, slow down
  - Packet loss is simplest form of feedback about congestion
- A simple congestion control algorithm to compute cwnd
  - Start with cwnd = 1 MSS
  - Initially, ramp up cwnd quickly, double cwnd every RTT (slow start)
  - After a threshold, be more careful, increase cwnd by 1 MSS every RTT (additive increase)
  - 3 dupack, slow down, halve the value of cwnd (multiplicative decrease)
  - If timeout, restart from beginning
- Different TCP variants use different congestion control algorithms for different types of applications, networks
  - Approximate heuristics, no one best congestion control algorithm



Understanding congestion



- What happens inside a router?
  - Look up destination IP address of received datagram, find next hop and outgoing link
  - If outgoing link is busy, packet is queued up until it can be transmitted
- What is congestion?
  - Network is a pipeline of links, the slowest link becomes the bottleneck
  - Queue builds up at the head of the bottleneck link, in bottleneck router
  - If queue at bottleneck router overflows, packets are dropped
  - Different connections may have different bottleneck links

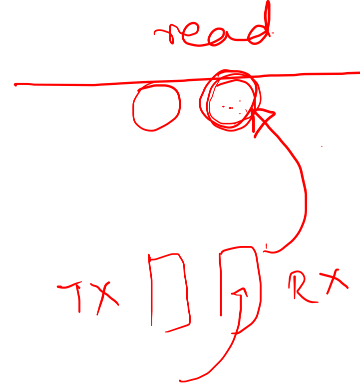


- Can we detect congestion before it causes packet drop?
  - Some routers can warn when queue starts to grow, before buffer fills up completely
  - When queue size crosses threshold, Random Early Detection (RED) routers drop packets with some probability, or set Explicit Congestion Notification (ECN) mark
  - ECN-aware TCP can use these warnings to adjust cwnd before packet drop

End-to-end delay in the network

- On every link in the network path, a packet experience delays
  - Transmission delay: time taken to put a packet onto the link
  - Propagation delay: time taken for the signal to reach the other end of link
  - Processing delay: time taken to process packet, look up forwarding table, ..
  - Queueing delay: time spent waiting in queue at router
- Round trip time RTT = sum of all delays for both data packet and ack
- BDP = bottleneck link bandwidth X RTT
- Varying network characteristics across different networks
  - Data center network paths have high bandwidth, low RTT (few milliseconds)
  - Internet-wide network paths have lower bandwidth, higher RTT (tens to hundreds of milliseconds)

Flow control



- What if network is fast, but receiver is slow?
  - If receiver OS is slow to handle interrupts, device RX ring will overflow, packet drop
  - If receiver application is slow to read from socket, socket RX queue will fill up
- TCP receiver indicates space left in socket RX queue in every ack (called receive window size)
- Sender sets window size to be minimum of cwnd, receive window size
- Flow control: sender slows down in order to not overwhelm receiver
  - Different reason for slowing down as compared to congestion control
- Ideally, receiver must set socket RX queue size to be at least equal to BDP, so that receive window is not reason for low throughput
- Network tools (e.g., iperf) run client and server on two hosts and report end to end TCP throughput achieved
  - If TCP throughput is low, but network is uncongested, can increase RX buffer size

Summary

- In this lecture:
  - Transport protocols
  - Mechanisms for reliability, congestion control, flow control
- Measure TCP throughput between two hosts (using tools like iperf). Change receive window size and observe impact on TCP throughput.