CS162
Operating Systems and Systems Programming
Lecture 11

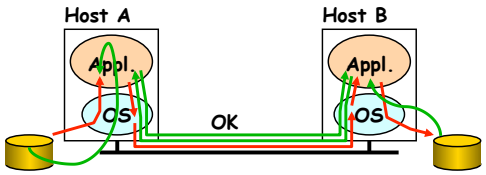
Reliability, Transport Protocols

October 5, 2011
Anthony D. Joseph and Ion Stoica
http://inst.eecs.berkeley.edu/~cs162

Goals for Today

- Finish e2e argument & fate sharing
- Transport: TCP/UDP
  - Reliability
  - Flow control

Example: Reliable File Transfer



- Solution 1: make each step reliable, and then concatenate them
- Solution 2: end-to-end **check** and try again if necessary

Discussion

- Solution 1 is incomplete
  - What happens if memory is corrupted?
  - Receiver has to do the check anyway!
- Solution 2 is complete
  - Full functionality can be entirely implemented at application layer with no need for reliability from lower layers
- *Is there any need to implement reliability at lower layers?*
  - Well, it could be more efficient

Summary of End-to-End Principle

Implementing this functionality in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity
- Probably imposes delay and overhead on all applications, even if they don't need functionality
- However, implementing in network can enhance performance in some cases
  - E.g., very losy link

Conservative Interpretation of E2E

- Don't implement a function at the lower levels of the system unless it can be completely implemented at this level
- Unless you can relieve the burden from hosts, don't bother

Moderate Interpretation

- Think twice before implementing functionality in the network
- If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement
- But do so only if it does not impose burden on applications that do not require that functionality

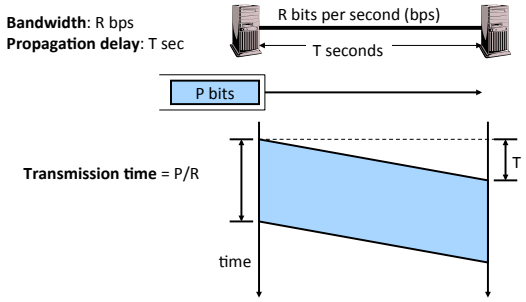
Related Notion of *Fate-Sharing*

- Idea: when storing state in a distributed system, keep it co-located with the entities that ultimately rely on the state
- Fate-sharing is a technique for dealing with failure
  - Only way that failure can cause loss of the critical state is if the entity that cares about it also fails ...
  - … in which case it doesn't matter
- Often argues for keeping *network state* at end hosts rather than inside routers
  - In keeping with End-to-End principle
  - E.g., packet-switching rather than circuit-switching
  - E.g., NFS file handles, HTTP "cookies"

Background: Definitions

- **Link bandwidth (capacity):** maximum rate (in bps) at which the sender can send data along the link
- **Propagation delay:** time it takes the signal to travel from source to destination
  - **Round Trip Time (RTT):** time it takes the signal to travel from source to destination and back
- **Packet transmission time:** time it takes the sender to transmit all bits of the packet

Background: Sending One Packet

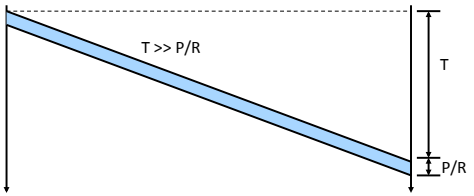


Propagation delay =T = Length/speed

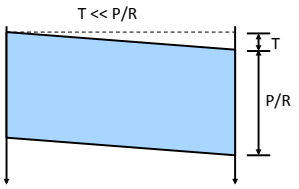
1m/speed = 3.3 usec in free space
4 usec in copper
5 usec in fiber

Sending one Packet: Examples

P = 1 Kbyte
R = 1 Gbps
100 Km, fiber =>
T = 500 usec
P/R = 1KB/1Gbps
= 8000b/10^9bps
= 10^{-6}sec = 8 usec



P = 1 Kbyte
R = 100 Mbps
1 Km, fiber =>
T = 5 usec
P/R = 80 usec



Reliable Transfer

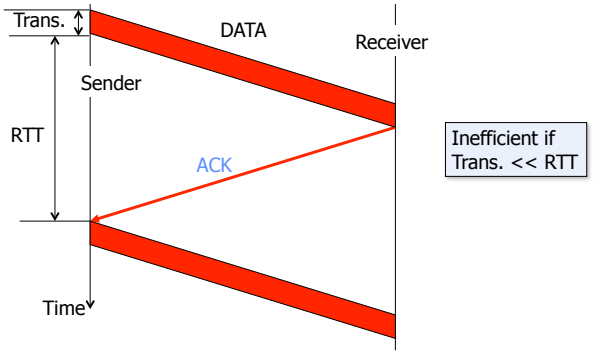
- Retransmit missing packets
  - Numbering of packets and ACKs
- Do this efficiently
  - Keep transmitting whenever possible
  - Detect missing packets and retransmit quickly
- Two schemes
  - Stop & Wait
  - Sliding Window (Go-back-n and Selective Repeat)

Detecting Packet Loss?

- Timeouts
  - Sender timeouts on not receiving ACK
- Missing ACKs
  - Sender ACKs each packet
  - Receiver detects a missing packet when seeing a gap in the sequence of ACKs
  - Need to be careful! Packets and acks might be reordered
- NACK: Negative ACK
  - Receiver sends a NACK specifying a packet its missing

Stop & Wait

- Send; wait for ack
- If timeout, retransmit; else repeat



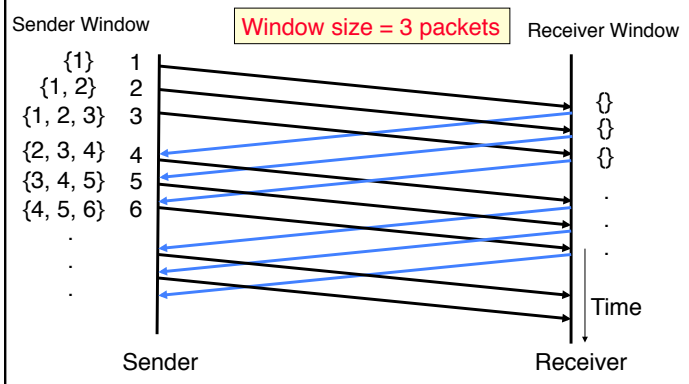
Sliding Window

- *window* = set of adjacent sequence numbers
- The size of the set is the *window size*
- Assume window size is n
- Let A be the last ack'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}
- Sender can send packets in its window
- Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}
- Receiver can accept out of sequence, if in window

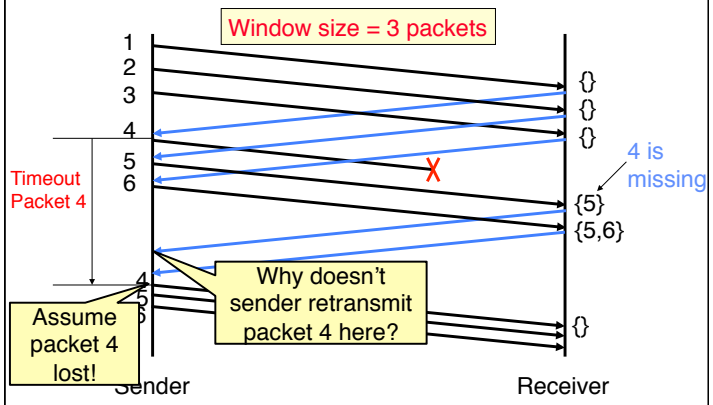
Go-Back-n (GBN)

- Transmit up to *n* unacknowledged packets
- If timeout for ACK(*k*), retransmit *k*, *k+1*, …

GBN Example w/o Errors



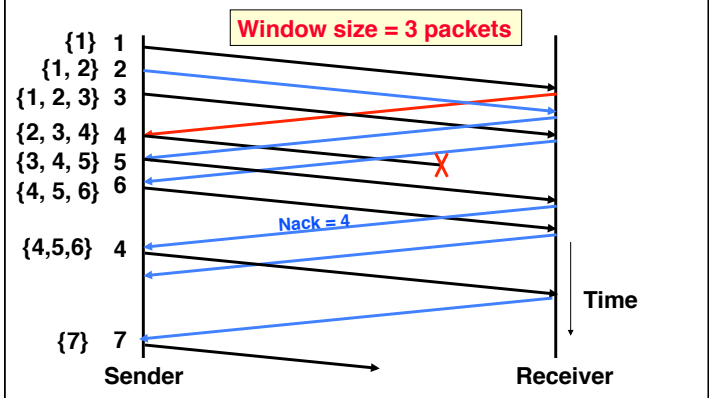
GBN Example with Errors



Selective Repeat (SR)

- Sender: transmit up to *n* unacknowledged packets;
- Assume packet *k* is lost
- Receiver: indicate packet *k* is missing
- Sender: retransmit packet *k*

SR Example with Errors



Observations

- With sliding windows, it is possible to fully utilize a link, provided the window size is large enough. Throughput is ~ (n/RTT)
  - Stop & Wait is like n = 1.
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits

Announcements

- Project 1 deadlines:
  - Code: Thursday, October 6, 11:59pm
  - Group evaluations: Friday, October 7, 11:59pm

5 Minute Break

Questions Before We Proceed?

Motivation for Transport Protocols

- IP provides a weak, but efficient service model (*best-effort*)
  - Packets can be delayed, dropped, reordered, duplicated
  - Packets have limited size (why?)
- IP packets are addressed to a host
  - How to decide which application gets which packets?
- How should hosts send packets into the network?
  - Too fast may overwhelm the network
  - Too slow is not efficient

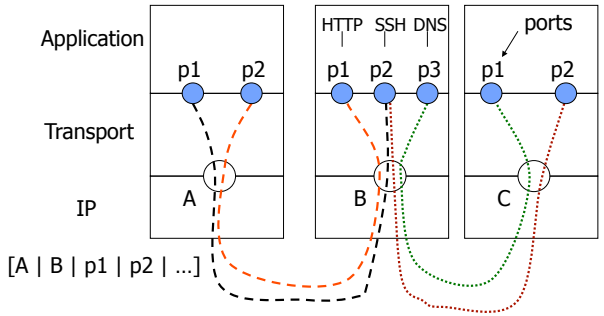
Transport Layer

- Provide a way to decide which packets go to which applications (*multiplexing/demultiplexing*)
- Can
  - Provide reliability, in order delivery, at most once delivery
  - Support messages of arbitrary length
  - Govern when hosts should send data → can implement congestion and flow control

Congestion vs. Flow Control

- Flow Control – avoid overflowing the receiver
- Congestion Control – avoid congesting the network
- What is network congestion?

Transport Layer (cont'd)



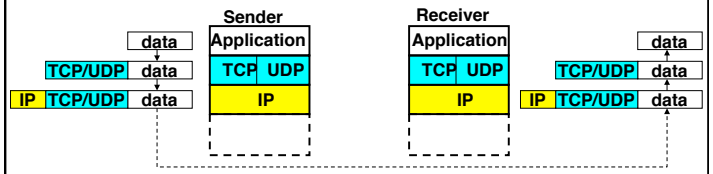
UDP: Not reliable
TCP: Ordered, reliable, well-paced

Ports

- Need to decide which application gets which packets
- Solution: map each socket to a *port*
- Client must know server's port
- Separate 16-bit port address space for UDP and TCP
  - (src_IP, src_port, dst_IP, dst_port) uniquely identifies TCP connection
- *Well known ports* (0-1023): everyone agrees which services run on these ports
  - e.g., ssh:22, http:80
  - On UNIX, must be root to gain access to these ports (why?)
- Ephemeral ports (most 1024-65535): given to clients
  - e.g. chat clients, p2p networks

Headers

- IP header → used for IP routing, fragmentation, error detection
- UDP header → used for multiplexing/demultiplexing, error detection
- TCP header → used for multiplexing/demultiplexing, flow and congestion control



UDP: User (Unreliable) Data Protocol

- Minimalist transport protocol
- Same best-effort service model as IP
- Messages up to 64KB
- Provides multiplexing/demultiplexing to IP
- Does not provide flow and congestion control
- Application examples: video/audio streaming

UDP Service & Header

- Service:
  - Send datagram from (IPa, Port1) to (IPb, Port2)
  - Service is unreliable, but error detection possible
- Header:

0	16	31
Source port	Destination port	
UDP length	UDP checksum	
Payload (variable)		

•UDP length is UDP packet length
(including UDP header and payload, but not IP header)
•Optional UDP checksum is over UDP packet

TCP: Transport Control Protocol

- Reliable, in-order, and at most once delivery
- Stream oriented: messages can be of arbitrary length
- Provides multiplexing/demultiplexing to IP
- Provides congestion control and avoidance
- Application examples: file transfer, chat

TCP Service

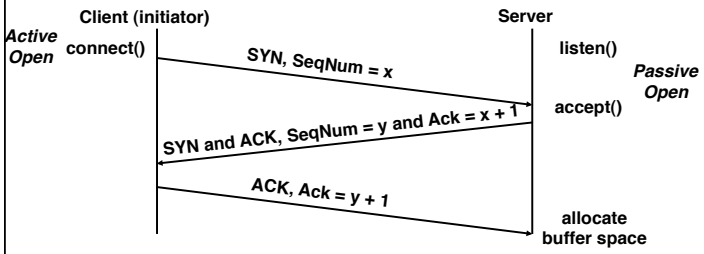
1) Open connection: 3-way handshaking

2) Reliable byte stream transfer from (IPa, TCP_Port1) to (IPb, TCP_Port2)
   - Indication if connection fails: Reset

3) Close (tear-down) connection

Open Connection: 3-Way Handshaking

- Goal: agree on a set of parameters: the start sequence number for each side
  - Starting sequence numbers are random

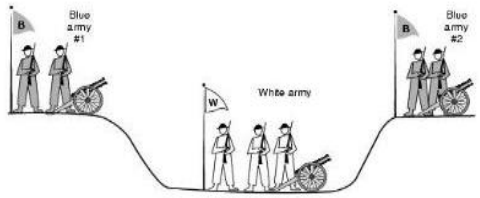


3-Way Handshaking (cont'd)

- Three-way handshake adds 1 RTT delay
- Why?
  - Congestion control: SYN (40 byte) acts as cheap probe
  - Protects against delayed packets from other connection (would confuse receiver)

Close Connection (Two Generals Problem)

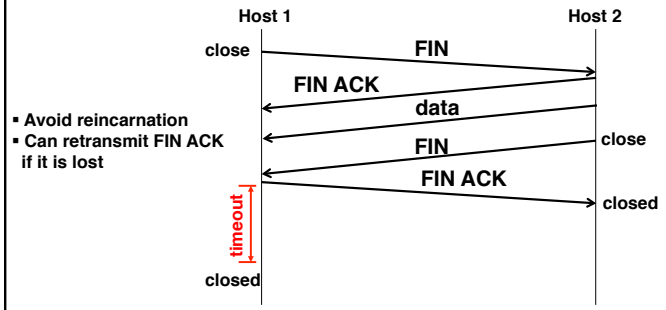
- Goal: both sides agree to close the connection
- Two-army problem:
  - "Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers."



- What is the solution?

Close Connection

- 4-ways tear down connection



Summary

- Reliable transmission
  - S&W not efficient → Go-Back-n
  - What to ACK? (cumulative, …)
- UDP: Multiplex, detect errors
- TCP: Reliable Byte Stream
  - 3-way handshaking