Computer Network Fundamentals
Spring 2010

Midterm Review
Andreas Terzis
Midterm

• Thursday 3/11 in class
• Closed book
• Mix of quantitative and problem solving questions
• Material
  – Everything up to and including Week 5
Moving data through the network

- Multiple end-hosts need to share the same upstream link
- How do you share this common resource?
Circuit Switching

• Divide link into smaller ‘sub-links’ and allocate each sub-link to an active network flow
  – Dedicated resources: no sharing
  – Guaranteed performance
  – Setup required
Packet switching

- Sequence of A&B packets share bandwidth on demand: **statistical multiplexing**
Store and Forward

• Takes L/R seconds to transmit a packet of L bits on link of R bps
• Store and Forward: entire packet must arrive at router before it can be transmitted on the next link
  – Delay = 3L/R (assuming zero propagation delay)
Four sources of packet delay (1)

1. Nodal processing
   - check bit errors
   - determine output link

2. Queueing
   - time waiting at output link for transmission
   - depends on congestion level of router
Four sources of packet delay (II)

3. Transmission delay:
   • $R =$ link bandwidth (bps)
   • $L =$ packet length (bits)
   • time to send bits into link = $L/R$

4. Propagation delay:
   • $d =$ length of physical link
   • $s =$ propagation speed in medium (~$2 \times 10^8$ m/sec)
   • propagation delay = $d/s$

Note: $s$ and $R$ are very different quantities!
Packet loss

- queue (aka buffer) preceding link in buffer has finite capacity
- packet arriving to full queue dropped (aka lost)
- lost packet may be retransmitted by previous node, by source end system, or not at all
Throughput

- **throughput**: rate (bits/time unit) at which bits transferred between sender/receiver
  - *instantaneous*: rate at given point in time
  - *average*: rate over longer period of time

![Diagram of throughput](image)

- Server sends bits (fluid) into pipe
- Pipe that can carry fluid at rate $R_s$ bits/sec
- Pipe that can carry fluid at rate $R_c$ bits/sec
Network Architecture

Networks are complex!
- many “pieces”:
  - hosts
  - routers
  - links of various media
  - applications
  - protocols
  - hardware, software

**Question:**
Is there any hope of organizing structure of network?

Or at least our discussion of networks?
Layering

- Layering is a particular form of modularization
- The system is broken into a vertical hierarchy of logically distinct entities (layers)
- The service provided by one layer is based solely on the service provided by layer below
- Rigid structure: easy reuse, performance suffers
Encapsulation
Application architectures

• Client-server
  – Including data centers / cloud computing
• Peer-to-peer (P2P)
• Hybrid of client-server and P2P
Client-server architecture

client/server

server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

Highly scalable but difficult to manage
HTTP overview

HTTP: hypertext transfer protocol

- Web’s application layer protocol
- client/server model
  - client: browser that requests, receives, “displays” Web objects
  - server: Web server sends objects in response to requests
HTTP overview (continued)

Uses TCP:
- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is “stateless”
- server maintains no information about past client requests
  aside

Protocols that maintain “state” are complex!
- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled
DNS: Domain Name System

People: many identifiers:
- SSN, name, passport#

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- “name”, e.g., ww.yahoo.com - used by humans

Q: map between IP addresses and name?

Domain Name System:
- distributed database implemented in hierarchy of many name servers
- application-layer protocol host, routers, name servers to communicate to resolve names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”
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.
## Multiplexing/demultiplexing

**Demultiplexing at rcv host:**
- Delivering received segments to correct socket

**Multiplexing at send host:**
- Gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

<table>
<thead>
<tr>
<th>Host 1</th>
<th>Host 2</th>
<th>Host 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application: P3</td>
<td>Application: P1</td>
<td>Application: P2</td>
</tr>
<tr>
<td>Transport</td>
<td>Transport</td>
<td>Transport</td>
</tr>
<tr>
<td>Network</td>
<td>Network</td>
<td>Network</td>
</tr>
<tr>
<td>Link</td>
<td>Link</td>
<td>Link</td>
</tr>
<tr>
<td>Physical</td>
<td>Physical</td>
<td>Physical</td>
</tr>
</tbody>
</table>

= socket  = process

Delivering received segments to correct socket.

Multiplexing at send host:
- Gathering data from multiple sockets, enveloping data with header (later used for demultiplexing).

**Diagrams:**

- **Demultiplexing at rcv host:**
  - Delivering received segments to correct socket

- **Multiplexing at send host:**
  - Gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)
UDP: User Datagram Protocol [RFC 768]

- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
  - lost
  - delivered out of order to app
- connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired
Reliable data transfer: channels with errors and loss

Network service: underlying channel can also lose packets (data or ACKs)
- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

**Approach:** sender waits “reasonable” amount of time for ACK
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #’s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer
Pipelining: increased utilization

\[ U_{\text{sender}} = \frac{3 \times L / R}{\text{RTT} + L / R} = \frac{0.24}{30.008} = 0.0008 \]

Increase utilization by a factor of 3!
Pipelining Protocols

**Go-back-N: overview**
- **sender**: up to N unACKed packets in pipeline
- **receiver**: only sends cumulative ACKs
  - doesn’t ACK pkt if there’s a gap
- **sender**: has timer for oldest unACKed pkt
  - if timer expires: retransmit all unACKed packets

**Selective Repeat: overview**
- **sender**: up to N unACKed packets in pipeline
- **receiver**: ACKs individual pkts
- **sender**: maintains timer for each unACKed pkt
  - if timer expires: retransmit only unACKed packet
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- **point-to-point:**
  - one sender, one receiver
- **reliable, in-order byte stream:**
  - no “message boundaries”
- **pipelined:**
  - TCP congestion and flow control set window size
- **send & receive buffers**

- **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- **connection-oriented:**
  - handshaking (exchange of control msgs) init’s sender, receiver state before data exchange
- **flow controlled:**
  - sender will not overwhelm receiver
TCP seq. #’s and ACKs

Seq. #’s:
- byte stream “number” of first byte in segment’s data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK

Host A
- Seq=42, ACK=79, data = ‘C’
- Seq=79, ACK=43, data = ‘C’
- Seq=43, ACK=80

Host B
- host ACKs receipt of ‘C’, echoes back ‘C’

simple telnet scenario
TCP reliable data transfer

- TCP creates rdt service on top of IP’s unreliable service
- pipelined segments
- cumulative ACKs
- TCP uses single retransmission timer

- retransmissions are triggered by:
  - timeout events
  - duplicate ACKs

- initially consider simplified TCP sender:
  - ignore duplicate ACKs
  - ignore flow control, congestion control
TCP Flow control: how it works

(suppose TCP receiver discards out-of-order segments)

• unused buffer space:
  = \text{rwnd}
  = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]

• receiver: advertises unused buffer space by including \text{rwnd} value in segment header

• sender: limits # of unACKed bytes to \text{rwnd}
  – guarantees receiver’s buffer doesn’t overflow
TCP Connection Management

TCP client lifecycle

TCP server lifecycle
TCP Congestion Control

- when \( cwnd < ssthresh \), sender in slow-start phase, window grows exponentially.
- when \( cwnd \geq ssthresh \), sender is in congestion-avoidance phase, window grows linearly.
- when triple duplicate ACK occurs, \( ssthresh \) set to \( cwnd/2 \), \( cwnd \) set to ~ \( ssthresh \)
- when timeout occurs, \( ssthresh \) set to \( cwnd/2 \), \( cwnd \) set to 1 MSS.
Why is TCP fair?

Two competing sessions:

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