Welcome to

**CS 3516:**
Advanced Computer Networks

Prof. Yanhua Li

Time: 9:00am – 9:50am M, T, R, and F
Location: Fuller 320
Fall 2016 A-term
Chapter 3: Transport Layer

our goals:

- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control

- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control
Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
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 vs. network layer

- **network layer:** logical communication between hosts
- **transport layer:** logical communication between processes
  - relies on, enhances, network layer services

**household analogy:**

12 kids in Ann’s house sending letters to 12 kids in Bill’s house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service
Internet transport-layer protocols

- **reliable, in-order delivery (TCP)**
  - congestion control
  - flow control
  - connection setup
- **unreliable, unordered delivery: UDP**
  - no-frills extension of “best-effort” IP
- **services not available:**
  - delay guarantees
  - bandwidth guarantees
Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
Multiplexing/demultiplexing

*Multiplexing at sender:* handle data from multiple sockets, add transport header (later used for demultiplexing)

*Demultiplexing at receiver:* use header info to deliver received segments to correct socket
Connectionless demultiplexing

- when host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

  IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest
Connectionless demux: example

```java
DatagramSocket serverSocket = new DatagramSocket(6428);
DatagramSocket mySocket1 = new DatagramSocket(5775);
DatagramSocket mySocket2 = new DatagramSocket(9157);
```

source port: 6428
dest port: 9157

source port: 9157
dest port: 6428

source port: ?
dest port: ?

source port: ?
dest port: ?

Transport Layer 3-10
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number

- demux: receiver uses all four values to direct segment to appropriate socket

- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple

- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request
Connection-oriented demux: example

Three segments, all destined to IP address: B, dest port: 80 are demultiplexed to different sockets.

Transport Layer 3-12
Connection-oriented demux: example

Transport Layer 3-13
Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
UDP: User Datagram Protocol [RFC 768]

- “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

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS

- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!
**UDP: segment header**

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
</tbody>
</table>

application data (payload)

**why is there a UDP?**

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired
**UDP checksum**

**Goal:** detect “errors” (e.g., flipped bits) in transmitted segment

**sender:**
- treat segment contents, including header fields, as sequence of 16-bit integers
- **checksum:** addition (one’s complement sum) of segment contents
- sender puts checksum value into UDP checksum field

**receiver:**
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected.
  
  *But maybe errors nonetheless? More later ....*
Internet checksum: example

example: add two 16-bit integers

\[
\begin{array}{cccccccccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
\text{wraparound} & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\text{sum} & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
\text{checksum} & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result
Quiz 5 on Tue (9/20)

1. Reliable transfer via UDP
2. UDP Checksum

Mid-term next Friday (Tentative)
Questions?