Chapter 3
Transport Layer

A note on the use of these Powerpoint slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we’d like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2016
© J.F Kurose and K.W. Ross, All Rights Reserved
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
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
   • segment structure
   • reliable data transfer
   • flow control
   • connection management
3.6 principles of congestion control
3.7 TCP congestion control
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
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
   • segment structure
   • reliable data transfer
   • flow control
   • connection management
3.6 principles of congestion control
3.7 TCP congestion control
**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

![Diagram showing multiplexing and demultiplexing](image-url)
How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number

- host uses *IP addresses & port numbers* to direct segment to appropriate socket

TCP/UDP segment format

- 32 bits
- source port #
- dest port #
- other header fields
- application data (payload)
- TCP/UDP segment format
Connectionless demultiplexing

- **recall**: created socket has host-local port #:
  
  ```java
  DatagramSocket mySocket1 = new DatagramSocket(12534);  
  ```

- **recall**: when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

- 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

DatagramSocket
serverSocket = new DatagramSocket (6428);

DatagramSocket
mySocket2 = new DatagramSocket (9157);

DatagramSocket
mySocket1 = new DatagramSocket (5775);

Transport Layer 3-11
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

- Host: IP address A
  - Source IP, port: B,80, Dest IP, port: A,9157
  - Source IP, port: A,9157, Dest IP, port: B,80

- Host: IP address C
  - Source IP, port: C,5775, Dest IP, port: B,80
  - Source IP, port: C,9157, Dest IP, port: B,80

Three segments, all destined to IP address: B, dest port: 80 are demultiplexed to different sockets.
Connection-oriented demux: example

- Transport Layer

Connection diagram:
- Host A: IP address A
  - Source IP, port: B,80
  - Destination IP, port: A,9157
- Server: IP address B
  - Source IP, port: C,5775
  - Destination IP, port: B,80
- Host C: IP address C
  - Source IP, port: C,9157
  - Destination IP, port: B,80

Threaded server

Transport Layer 3-14
Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer
3.5 connection-oriented transport: TCP
   • segment structure
   • reliable data transfer
   • flow control
   • connection management
3.6 principles of congestion control
3.7 TCP congestion control
**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

**UDP use:**

- streaming multimedia apps (loss tolerant, rate sensitive)
- DNS
- SNMP

**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>

- **32 bits**

- **UDP segment format**

- **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

\[
\begin{array}{cccccccccccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
\]

wraparound \[1\] 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1

\[
\begin{array}{cccccccccccccccc}
\text{sum} & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
\text{checksum} & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
Chapter 3 outline

3.1 transport-layer services
3.2 multiplexing and demultiplexing
3.3 connectionless transport: UDP
3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control
3.7 TCP congestion control