Computer Networks

Transport Layer Services
Multiplexing/Demultiplexing

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 sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- 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
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)

transport layer services

P1

P2

P3

P4

host 1

host 2

host 3

Transport Layer Services

How demultiplexing works

• host receives IP datagrams
  – each datagram has source IP address, destination IP address
  – each datagram carries 1 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

<table>
<thead>
<tr>
<th></th>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>other header fields</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

application

data
(message)

Transport Layer Services
Connectionless demultiplexing

- Create sockets with port numbers:
  
  ```java
  DatagramSocket mySocket1 = new DatagramSocket(12534);
  DatagramSocket mySocket2 = new DatagramSocket(12535);
  ```

- UDP socket identified by two-tuple:
  
  (dest IP address, dest port number)

- When host receives UDP segment:
  
  - checks destination port number in segment
  - directs UDP segment to socket with that port number

- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

---

Connectionless demux (cont)

```java
DatagramSocket serverSocket = new DatagramSocket(6428);
```
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- receiving host 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 (cont)
Connection-oriented demux: Threaded Web Server