Transport Layer

Dmitri Loguinov
Texas A&M University

February 23, 2024
Chapter 3: Transport Layer

Our goals:

• Understand principles behind transport layer services:
  – Multiplexing/demultiplexing
  – Reliable data transfer
  – Flow control
  – Congestion control

• Learn about transport layer protocols in the Internet:
  – UDP: connectionless transport
  – TCP: connection-oriented transport
Chapter 3: Roadmap

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

- **Transport layer**: logical communication between processes on different hosts
  - Relies on and enhances network-layer services
- **Network layer**: logical communication between hosts
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 or loss guarantees
  - Bandwidth guarantees
Chapter 3: Roadmap

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

Demultiplexing at receiver host:
Delivering received segments to correct socket

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

= socket = process

<table>
<thead>
<tr>
<th>application</th>
<th>P3</th>
<th>transport</th>
<th>network</th>
<th>link</th>
<th>physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>host 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>application</th>
<th>P1</th>
<th>transport</th>
<th>network</th>
<th>link</th>
<th>physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>host 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| P2 | application | transport | network | link | physical |
|    |             |           |         |      |          |
| host 3 |           |           |         |      |          |
How Demultiplexing Works

- Host receives IP datagrams
  - Each datagram has source IP address and destination IP address
- Each datagram carries one transport-layer header
  - Transport header starts with source and destination port numbers
- Kernel uses port numbers to direct packets to appropriate socket or reject the message
  - Each port # is a 16-bit unsigned integer (1-65535)
Connectionless Demultiplexing

- Create a SOCK_DGRAM socket
- **Bind** the socket
  - Server: specify a well-known port (e.g., 53 for DNS)
  - Client: bind to port 0 (OS assigns next available #)
- Use sendto(), recvfrom()
- Target UDP socket is identified by a 2-tuple: (dest IP address, dest port number)

- When host receives UDP segment:
  - OS checks destination port/IP in segment
  - Directs segment to the socket with a matching combination if socket is open; rejects otherwise

- IP datagrams with different source IP addresses and/or source port numbers may be directed to the same socket!
Connectionless Demultiplexing (Cont)

SP = source port, DP = destination port

SP provides “return address”
Connection-Oriented Demultiplexing

- TCP socket identified by a 4-tuple:
  - Source IP address
  - Source port number
  - Destination IP address
  - Destination port number

- Receiver host uses all four values to find appropriate socket

- **Clients**: each socket must have unique port

- **Servers**: possible to have multiple TCP sockets with same port number:
  - Each socket identified by its own 4-tuple

- Web servers have different sockets for each connecting client
  - All are on port 80
  - Non-persistent HTTP may have different socket for each request
Connection-Oriented Demultiplexing (Cont)

Web server spawns a new process per connection

port 80

SP = source port, DP = destination port;
S-IP = source IP, D-IP = destination IP
Connection-Oriented Demultiplexing (Cont)

Web server spawns a new thread per connection

SP = source port, DP = destination port;
S-IP = source IP, D-IP = destination IP