The Transport Layer
Multiplexing, Error Detection, & UDP

Jasleen Kaur
February 21, 2019
Transport Layer Protocols

Internet transport services

- **TCP**: Reliable, in-order, unicast delivery
  - Congestion control
  - Flow control
  - Connection setup

- **UDP**: Unreliable, unordered ("best-effort"); unicast or multicast delivery
  - (Minimal) error detection

- Services not available:
  - Performance guarantees
    - No guarantees of available bandwidth
    - No guarantees of end-to-end delay
  - Other (non-unicast) delivery models
    - Multicast (reliable v. unreliable)
    - Anycast

Transport Layer Protocols & Services

Outline

- Fundamental transport layer services
  - Multiplexing/Demultiplexing
  - Error detection
  - Reliable data delivery
  - Pipelining
  - Flow control
  - Congestion control

- Service implementation in Internet transport protocols
  - UDP
  - TCP
Fundamental Transport Layer Services

Multiplexing/Demultiplexing

- Each end-system has a single protocol “stack”
  - The stack is shared between all applications using the network

- Multiplexing is the process of allowing multiple applications to use the network simultaneously
  - (To send data into the network concurrently)

- Demultiplexing is the process of delivering received data to the appropriate application

Multiplexing/Demultiplexing

Review: Protocol layering in the Internet

- At the sender, each layer takes data from above
  - May subdivide into multiple data units at sending layer
  - Adds header information to create new data unit
  - Passes new data unit to layer below
- The process is reversed at the receiver
Multiplexing/Demultiplexing

Multiplexing

- Gathering data from multiple application processes, enveloping data with header (later used for demultiplexing)

- Based on IP addresses and sender and receiver port numbers
  - Source and destination port numbers carried in each segment
  - (Recall: well-known port numbers for specific applications)

TCP/UDP segment format

Demultiplexing

Demultiplexing is the process of delivering received segments to the correct application-layer process

- IP address (in network-layer datagram header) identifies the receiving machine
- Port number (in transport-layer segment header) identifies the receiving process
Multiplexing/Demultiplexing

Transport protocol specific demultiplexing

- Demultiplexing actions depend on whether the transport layer is connectionless (UDP) or connection-oriented (TCP)

- UDP demultiplexes segments to the *socket*
  - UDP uses 2-tuple
    - \(<destination\ IP\ addr,\ destination\ port\ nbr>\)
    - to identify the socket
  - Socket is “owned” by some process (allocated by OS).

- TCP demultiplexes segments to the *connection*
  - TCP uses 4-tuple
    - \(<source\ IP\ addr,\ source\ port\ nbr,\ destination\ IP\ addr,\ destination\ port\ nbr>\)
    - to identify connection
  - Connection (and its socket) is owned by some process

Examples

Web client

Host A

source IP: A

dest. port: x

source IP: C

dest IP: B

dest. port: 80

dest. port: 80

Web server

host B

source IP: C

dest IP: B

dest. port: x

dest. port: 80

dest. port: 80

Web server port use (TCP)

DNS

Server B

source IP: C

dest IP: B

dest. port: 53

dest. port: 53

DNS server port use (UDP)

Web client Host C

source IP: C

dest IP: B

dest. port: 53

dest. port: 53

DNS Server B

source IP: C

dest IP: B

dest. port: 53

dest. port: 53

Web client Host A

source IP: A

dest IP: B

dest. port: x

dest. port: x

DNS server port use (UDP)
**Fundamental Transport Layer Services**

"Best Effort" Delivery

- Goal: Provide error detection and multiplexing but no delivery guarantees
  - The characteristics of the underlying network layer will determine the reliability of data delivery

**Internet Transport Protocols**

User Datagram Protocol (UDP) [RFC 768]

- No frills, “bare bones” Internet transport protocol
- Best effort service — UDP segments may be:
  - Lost
  - Delivered out of order to the application
  - Delivered multiple times to the application
- “Connectionless”
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

**UDP segment format**

<table>
<thead>
<tr>
<th>32 bits</th>
<th>source port #</th>
<th>dest. port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>application data (message payload)</td>
<td></td>
</tr>
<tr>
<td>checksum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Length field is length in bytes, of UDP segment (including header)
User Datagram Protocol (UDP)

Is unreliable, unordered communications useful?

- Who uses UDP?
  - Often used for streaming multimedia applications
  - Loss tolerant
  - Rate sensitive

- Other UDP uses (why?):
  - DNS
  - SNMP
  - Routing protocols

- Reliable transfer over UDP still possible
  - Reliability can always be added at the application layer
  - (Application-specific error recovery)

Why use 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

User Datagram Protocol (UDP)

Checksum computation

- The UDP checksum allows the receiver to detect errors in transmitted segment
  - Errors are “flipped” bits

- Sender computation:
  - Treat segment contents as a sequence of 16-bit integers
  - Sum the segment’s contents, place the 1’s complement of the sum into the checksum field

- Example:
  - Sum of segment = 1010101110011011
  - Checksum = 0101010001100100

"Theorem:"

\[
\text{segment sum + checksum} = 1111111111111111
\]
User Datagram Protocol (UDP)
Checksum computation

- Receiver computation:
  - Compute checksum of received segment (including received checksum)
  - Compare value to all 1’s
  - If equal — No error detected, segment “OK”
  - If not equal — Error detected, now what?!
    - Retransmit?
    - Discard?
    - Deliver?

![UDP segment format]

32 bits

<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 (message payload)

"Theorem:"

segment sum + checksum =

1111111111111111