RSC Part III: Transport Layer
1. Basic Concepts

Redes y Servicios de Comunicaciones
Universidad Carlos III de Madrid

These slides are, mainly, part of the companion slides to the book “Computer Networking: A Top Down Approach” generously made available by their authors (see copyright below). The slides have been adapted, where required, to the teaching needs of the subject above.

All material copyright 1996-2009
J.F Kurose and K.W. Ross. All Rights Reserved

Network Layer II-1

RSC Part III: Transport Layer

- III.1 Basic Transport layer concepts
  - Transport layer Principles
  - Transport layer Services
  - Multiplexing and Demultiplexing

- III.2 UDP
  - UDP Segment format
  - UDP checksum

- III.3 TCP
  - TCP connection
  - TCP Segment, sequence and ack numbers
  - RTT Estimation and Timeout
  - Reliable Data Transfer
  - Flow Control
  - TCP connection Management
  - TCP Congestion Control

Network Layer II-2
**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.
  - Receive 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

RSC Part III: Transport Layer

- III.1 Basic Transport layer concepts
  - Transport layer Principles
  - Transport layer Services
  - Multiplexing and Demultiplexing

- III.2 UDP
  - UDP Segment format
  - UDP checksum

- III.3 TCP
  - TCP connection
  - TCP Segment, sequence and ack numbers
  - RTT Estimation and Timeout
  - Reliable Data Transfer
  - Flow Control
  - TCP connection Management
  - TCP Congestion Control

Transport Layer 3-5

Network Layer II-6
**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)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>P1</td>
</tr>
<tr>
<td>transport</td>
<td></td>
</tr>
<tr>
<td>network</td>
<td></td>
</tr>
<tr>
<td>link</td>
<td></td>
</tr>
<tr>
<td>physical</td>
<td></td>
</tr>
</tbody>
</table>

= socket  = process

**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

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bits</td>
<td></td>
</tr>
</tbody>
</table>

TCP/UDP segment format

- Application data (message)

Transport Layer 3-7
Connectionless demultiplexing

- Create sockets with port numbers:
  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)

DatagramSocket serverSocket = new DatagramSocket(6428);

SP provides “return address”
Connection-oriented demux

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

- recv 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)

<table>
<thead>
<tr>
<th>Client IP: A</th>
<th>Server IP: C</th>
<th>Client IP: B</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P4, P5, P6</td>
<td>P2, P3</td>
</tr>
<tr>
<td>SP: 9157</td>
<td>SP: 9157</td>
<td>SP: 9157</td>
</tr>
<tr>
<td>DP: 80</td>
<td>DP: 80</td>
<td>DP: 80</td>
</tr>
<tr>
<td>S-IP: A</td>
<td>S-IP: B</td>
<td>S-IP: B</td>
</tr>
<tr>
<td>D-IP: C</td>
<td>D-IP: C</td>
<td>D-IP: C</td>
</tr>
</tbody>
</table>
Connection-oriented demux: Threaded Web Server

RSC Part III: Transport Layer

- III.1 Basic Transport layer concepts
  - Transport layer Principles
  - Transport layer Services
  - Multiplexing and Demultiplexing

- III.2 UDP
  - UDP Segment format
  - UDP checksum

- III.3 TCP
  - TCP connection
  - TCP Segment, sequence and ack numbers
  - RTT Estimation and Timeout
  - Reliable Data Transfer
  - Flow Control
  - TCP connection Management
  - 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

Why is there a 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

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!

<table>
<thead>
<tr>
<th>Source port #</th>
<th>Destination port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

UDP segment format

Application data (message)
UDP checksum

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

**Sender:**
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1’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

- **Note**
  - When adding numbers, a carryout from the most significant bit needs to be added to the result

- **Example:** add two 16-bit integers

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

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

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