Transport Layer (TCP/UDP)

Part A
Where we are in the Course

• Moving down to the Transport Layer!
The Transport Layer

• The transport layer provides *end-to-end* connectivity
• To the transport layer, its payload is just bytes
Encapsulation

*TCP Segment*

802.11 | IP | TCP | HTTP | HTTP payload

*IP Packet*

Frame

*UDP Datagram*

802.11 | IP | UDP | DNS header and payload
Transport Layer Services

• Provide different kinds of data delivery across the network to applications

<table>
<thead>
<tr>
<th></th>
<th>Unreliable</th>
<th>Reliable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packets</td>
<td>Datagrams (UDP)</td>
<td></td>
</tr>
<tr>
<td>Bytestream</td>
<td></td>
<td>Streams (TCP)</td>
</tr>
</tbody>
</table>

• *Could there be protocols in the two empty boxes?*
Comparison of Internet Transports: Function

<table>
<thead>
<tr>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streams</td>
<td>Datagrams</td>
</tr>
<tr>
<td>Connections</td>
<td>Connectionless</td>
</tr>
<tr>
<td><strong>Bytes are delivered to receiving app reliably</strong> (once, and in order)</td>
<td><strong>Packets may be lost, reordered, duplicated</strong> (but not corrupted)</td>
</tr>
<tr>
<td><strong>Arbitrary length content</strong></td>
<td><strong>Fixed maximum datagram size</strong></td>
</tr>
</tbody>
</table>
## Comparison of Internet Transports: Performance

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection latency</strong></td>
<td></td>
<td><strong>No delay</strong></td>
</tr>
<tr>
<td><strong>Segment delivery latency</strong> (&quot;nagling&quot;)</td>
<td>Datagram is sent now</td>
<td></td>
</tr>
<tr>
<td><strong>Flow control</strong></td>
<td>matches sender’s rate to receiver’s capability</td>
<td>No flow control (can lead to many lost datagrams)</td>
</tr>
<tr>
<td><strong>Congestion control</strong></td>
<td>matches sender’s rate to network’s capability</td>
<td>No congestion control (can lead to many lost datagrams)</td>
</tr>
</tbody>
</table>
Socket API

• Simple OS abstraction to use the network
  • The “network” API (really Transport service) used to write all Internet apps
  • Part of all major OSes and languages; originally Berkeley (Unix) ~1983

• Supports both Internet transport services (TCP and UDP)

• The OS provides sockets; the Internet provides the port abstraction
Socket API

- **Sockets** are associated with (“bound to”) Internet ports
## Socket API

Same API used for Streams and Datagrams

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKET</td>
<td>Create a new communication endpoint</td>
</tr>
<tr>
<td>BIND</td>
<td>Associate a local address (port) with a socket</td>
</tr>
<tr>
<td>LISTEN</td>
<td>Announce willingness to accept connections</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>Passively establish an incoming connection</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>SEND(TO)</td>
<td>Send some data over the socket</td>
</tr>
<tr>
<td>RECEIVE(FROM)</td>
<td>Receive some data over the socket</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Release the socket</td>
</tr>
</tbody>
</table>

Only needed for Streams

To/From for Datagrams

*Note: A language layer can obscure this interface*
Ports

• Application process is identified by the tuple <IP address, protocol, and port>
  • Ports are 16-bit integers representing local “mailboxes” that a process leases

• Servers often bind to “well-known ports”
  • numbered below 1024
  • requires administrative privileges (“privileged ports”)

• Clients often assigned “ephemeral” ports
  • Chosen by OS; change from run to run
Some Well-Known Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>20, 21</td>
<td>FTP</td>
<td>File transfer</td>
</tr>
<tr>
<td>22</td>
<td>SSH</td>
<td>Remote login</td>
</tr>
<tr>
<td>25</td>
<td>SMTP</td>
<td>Email</td>
</tr>
<tr>
<td>80</td>
<td>HTTP</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>110</td>
<td>POP-3</td>
<td>Remote email access</td>
</tr>
<tr>
<td>143</td>
<td>IMAP</td>
<td>Remote email access</td>
</tr>
<tr>
<td>443</td>
<td>HTTPS</td>
<td>Secure Web (HTTP over SSL/TLS)</td>
</tr>
<tr>
<td>543</td>
<td>RTSP</td>
<td>Media player control</td>
</tr>
<tr>
<td>631</td>
<td>IPP</td>
<td>Printer sharing</td>
</tr>
</tbody>
</table>
Topics

• Service models
  • Socket API and ports
  • Datagrams, Streams
• User Datagram Protocol (UDP)
• Reliable Transmission
• Sliding Window (TCP)
• Flow control (TCP)
• Retransmission timers (TCP)
• Congestion control (TCP)
UDP
User Datagram Protocol (UDP)

• Used by apps that don’t want TCP semantics or for which TCP performance characteristics are unacceptable
  • Voice-over-IP
  • DNS, RPC
  • DHCP

(If application wants reliability and messages then it has work to do!)
Example Protocol Over Datagram Sockets

Client (host 1)  Time  Server (host 2)

request

reply
Datagram Sockets

The protocol implied by this diagram is horribly broken!
UDP Header

• Uses ports to identify sending and receiving application processes
• Datagram length limited to 64K
• Checksum (16 bits) for reliability
UDP Header

• Optional checksum covers UDP segment and IP pseudoheader
  • Checks key IP fields (addresses)
  • Value of zero means “no checksum”
Internet Checksum

• Idea:
  • sender sums up data in N-bit words
    • results in a 16-bit value that is a function of the data
  • receiver performs same summation
  • if value receiver computes doesn’t match value sent by sender, the packet has been corrupted
  • Widely used in, e.g., TCP/IP/UDP

| 1500 bytes | 16 bits |
Internet Checksum

• Sum is defined in 1s complement arithmetic (must add back carries)
  • And it’s the negative sum

• “The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words ...” – RFC 791
Internet Checksum (2)

Sending:
1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

\begin{align*}
0001 & \rightarrow f204 \\
f4f5 & \rightarrow f6f7
\end{align*}
Internet Checksum (3)

Sending:
1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum
Internet Checksum (4)

Receiving:
1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
0001
f204
f4f5
f6f7
+ 220c
-----
```
Internet Checksum (5)

Receiving:
1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

\[
\begin{align*}
0001 & \quad f204 \\
04f5 & \quad f6f7 \\
& \quad + 220c \\
\hline
2fefd & \\
0000 & \quad + 0000 \\
\end{align*}
\]