Last Time

• We began on the Transport layer

• Focus
  – How do we send information reliably?

• Topics
  – ARQ and sliding windows
This Time

• More on the Transport Layer

• Focus
  – How do we connect processes?

• Topics
  – Naming processes
  – Connection setup / teardown
  – Flow control

Naming Processes/Services

• Process here is an abstract term for your Web browser (HTTP), Email servers (SMTP), hostname translation (DNS), RealAudio player (RTSP), etc.

• How do we identify for remote communication?
  – Process id or memory address are OS-specific and transient

• So TCP and UDP use Ports
  – 16-bit integers representing mailboxes that processes “rent”
  – Identify process uniquely as (IP address, protocol, port)
Picking Port Numbers

- We still have the problem of allocating port numbers
  - What port should a Web server use on host X?
  - To what port should you send to contact that Web server?

- Servers typically bind to “well-known” port numbers
  - e.g., HTTP 80, SMTP 25, DNS 53, … look in /etc/services
  - Ports below 1024 reserved for “well-known” services

- Clients use OS-assigned temporary (ephemeral) ports
  - Above 1024, recycled by OS when client finished

User Datagram Protocol (UDP)

- Provides message delivery between processes
  - Source port filled in by OS as message is sent
  - Destination port identifies UDP delivery queue at endpoint

<table>
<thead>
<tr>
<th>SrcPort</th>
<th>DstPort</th>
<th>Checksum</th>
<th>Length</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UDP Delivery

- Packets arrive
- Ports
- Message Queues
- DeMux
- Kernel boundary
- Application process
- Application process
- Application process

UDP Checksum

- UDP includes optional protection against errors
  - Checksum intended as an end-to-end check on delivery
  - So it covers data, UDP header, and IP pseudoheader
Transmission Control Protocol (TCP)

- Reliable bi-directional bytestream between processes
  - Message boundaries are not preserved

- Connections
  - Conversation between endpoints with beginning and end

- Flow control (later)
  - Prevents sender from over-running receiver buffers

- Congestion control (later)
  - Prevents sender from over-running network buffers

TCP Delivery

![TCP Delivery Diagram](image-url)
## TCP Header Format

- Ports plus IP addresses identify a connection

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td></td>
</tr>
<tr>
<td>DstPort</td>
<td></td>
</tr>
<tr>
<td>SequenceNum</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>0</td>
</tr>
<tr>
<td>Flags</td>
<td></td>
</tr>
<tr>
<td>AdvertisedWindow</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>UrgPtr</td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

- Sequence, Ack numbers used for the sliding window
  - Congestion control works by controlling the window size
TCP Header Format

- Flags may be URG, ACK, PSH, RST, SYN, FIN

```
0  4  10  16  31
SrcPort         DstPort
SequenceNum
Acknowledgment
HdrLen  0  Flags  AdvertisedWindow
Checksum        UrgPtr
Options (variable)
Data
```

TCP Header Format

- Advertised window is used for flow control

```
0  4  10  16  31
SrcPort         DstPort
SequenceNum
Acknowledgment
HdrLen  0  Flags  AdvertisedWindow
Checksum        UrgPtr
Options (variable)
Data
```
Other TCP Header Fields

- Header length allows for variable length TCP header with options for extensions such as timestamps, selective acknowledgements, etc.
- Checksum is analogous to that of UDP
- Urgent pointer/data not used in practice
- Very few bits not assigned ...

Connection Establishment

- Both sender and receiver must be ready before we start to transfer the data
  - Sender and receiver need to agree on a set of parameters
  - e.g., the Maximum Segment Size (MSS)
- This is signaling
  - It sets up state at the endpoints
  - Compare to “dialing” in the telephone network
- In TCP a Three-Way Handshake is used
Three-Way Handshake

- Opens both directions for transfer

![Diagram of Three-Way Handshake]

SYN, SequenceNum = x
SYN + ACK, SequenceNum = y
ACK, Acknowledgment = x + 1
ACK, Acknowledgment = y + 1
+data

Some Comments

- We could abbreviate this setup, but it was chosen to be robust, especially against delayed duplicates
  - Three-way handshake from Tomlinson 1975

- Choice of changing initial sequence numbers (ISNs) minimizes the chance of hosts that crash getting confused by a previous incarnation of a connection

- But with random ISN it actually proves that two hosts can communicate
  - Weak form of authentication
TCP State Transitions

Again, with States

Active participant (client)         Passive participant (server)

SYN_SENT  SYN_RCVD  ESTABLISHED  LISTEN

SYN, SequenceNum = x
SYN + ACK, SequenceNum = y,
Acknowledgment = x + 1
ACK, Acknowledgment = y + 1
+data
Connection Teardown

- Orderly release by sender and receiver when done
  - Delivers all pending data and “hangs up”

- Cleans up state in sender and receiver

- TCP provides a “symmetric” close
  - both sides shutdown independently

TCP Connection Teardown

![Diagram showing TCP connection teardown process]

- Web server
  - FIN_WAIT_1
  - FIN_WAIT_2
  - TIME_WAIT
  - CLOSED

- Web browser
  - CLOSE_WAIT
  - LAST_ACK
  - CLOSED
The TIME_WAIT State

- We wait 2MSL (two times the maximum segment lifetime of 60 seconds) before completing the close

- Why?

- ACK might have been lost and so FIN will be resent
- Could interfere with a subsequent connection

Key Concepts

- We use ports to name processes in TCP/UDP
  - “Well-known” ports are used for popular services
- Connection setup and teardown complicated by the effects of the network on messages
  - TCP uses a three-way handshake to set up a connection
  - TCP uses a symmetric disconnect