Last Class

- We talked about how to implement a reliable channel in the transport layer
- Approaches
  - ARQ (Automatic Repeat reQuest), Sliding window
    - Good RTT estimates
    - Packet sequencing as an indicator of loss (Fast Retransmit)
  - FEC (Forward Error Correction)
    - Redundant data encoding
    - Appropriate for asymmetric channels, multicast, or high delay/high loss channels

Today

- Finish basic transport protocol issues in context of
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
- Connection-oriented vs connection-less transport
  - Naming
  - Connection setup
  - Connection teardown
- Flow control
  - How do we manage buffering at the endpoints?

Naming Processes/Services

- Process here is an abstract term for your Web browser (HTTP), Email servers (SMTP), hostname translation (DNS), RealAudio player (RTSP/RDT), 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 traditionally reserved for “well-known” services
- Clients use OS-assigned temporary (ephemeral) ports
  - Above 1024, recycled by OS when client finished

Transmission Control Protocol (TCP)

- Reliable bi-directional byte stream between processes
  - Message boundaries are not preserved
- Connection-oriented
  - Conversation between two endpoints with beginning and end
  - Flow control (later)
    - Prevents sender from over-running receiver buffers
  - Congestion control (next class)
    - Prevents sender from over-running network buffers
TCP Delivery

- Application process
  - Write bytes
  - TCP
    - Send buffer
    - Segment
    - Transmit segments
  - TCP
    - Receive buffer
  - Read bytes

TCP Delivery

- Application process
  - Write bytes
  - TCP
    - Send buffer
    - Segment
    - Transmit segments
  - TCP
    - Receive buffer
  - Read bytes

TCP Header Format

- Sequence, Ack numbers used for the sliding window
- How big a window? Flow control/congestion control determine

TCP Header Format

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

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
  - Most important: sequence number space in each direction
  - Lots of other parameters: e.g., the Maximum Segment Size

- This is signaling
  - It sets up state at the endpoints
  - Similar to “dialing” in the telephone network

- Handshake protocols: setup state between two oblivious endpoints

Two-way handshake?

- Active participant (client)
  - SYN, SequenceNum = x
  - +data

- Passive participant (server)
  - SYN, SequenceNum = y
  - What’s wrong here?
Two-way handshake?

- Active participant (client)
- Passive participant (server)
- Old SYN, SequenceNum = x
- New SYN, SequenceNum = y
- Delayed old SYN
- SYN, SequenceNum = y
- +data
- Rejected

Three-Way Handshake

- Opens both directions for transfer
- Active participant (client)
- Passive participant (server)
- SYN, SequenceNum = x
- SYN + ACK, SequenceNum = y
- 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
- How to choose ISNs?
  - Maximize period between reuse
  - Minimize ability to guess (why?)

TCP State Transitions

- CLOSED
- LISTEN
- SYN_RCVD
- SYN_SENT
- ESTABLISHED
- FIN_WAIT_1
- FIN_WAIT_2
- CLOSE_WAIT
- LAST_ACK
- CLOSING
- TIME_WAIT
- Active open/SYN
- SYN/SYN + ACK
- SYN + ACK/ACK
- FIN/ACK
- CLOSE
- Timeout after two segment lifetimes

Again, with States

- Active participant (client)
- Passive participant (server)
- SYN_SENT
- LISTEN
- SYN_RCVD
- SYN_SENT
- ESTABLISHED
- +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

The TIME_WAIT State

Flow Control

TCP Header Format

Sender and Receiver Buffering

Example – Exchange of Packets
Example – Buffer at Sender

| T=1 | 1 2 3 4 5 6 7 8 9 |
| T=2 | 1 2 3 4 5 6 7 8 9 |
| T=3 | 1 2 3 4 5 6 7 8 9 |
| T=4 | 1 2 3 4 5 6 7 8 9 |
| T=5 | 1 2 3 4 5 6 7 8 9 |
| T=6 | 1 2 3 4 5 6 7 8 9 |

Lots of icky details
- Window probes
- Silly Window Syndrome
- Nagel's algorithm
- PAWS
- Etc...
- Steven’s books “TCP/IP Illustrated (vol 1,2)” is a great source of information on this

Example Icky Detail: Advertised Window Deadlock
- If the receiving process does not empty the buffer (e.g., not scheduled), then the sender fills up the receiver’s buffer
  - Advertised Window is 0
  - Effective Window goes to 0 when all data is ACKed
- Problem: When can the sender start sending again?
  - No timeouts because all data is ACKed
  - No packets from receiver with a new Advertised Window because receiver isn’t running
- Solution: Ping with a segment of 1 byte of data
  - Eventually receiver responds with a new Advert. Window

Misc TCP Header fields
- Header length allows for variable length TCP header with options for extensions such as timestamps, selective acknowledgements, etc.
- Checksum protects TCP header and data
- Urgent pointer/data not used in practice

TCP applications
- HTTP/WWW
- FTP
- SMTP, POP, IMAP (E-mail)
- Why is TCP well suited to these applications?

User Datagram Protocol (UDP)
- Provides unreliable message delivery between processes
  - Source port filled in by OS as message is sent
  - Destination port identifies UDP delivery queue at endpoint
- Connectionless (no state about who talks to whom)
**UDP Delivery**

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

**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

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

**Applications for UDP**

- Streaming media
- DNS (Domain Name Service)
- NTP (Network Time Protocol)
- Why is UDP appropriate for these?

**Homework**

- Next class: Congestion Control
- No new reading, make sure you’re caught up