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** bytestream 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

Transmit segments
Segment  Segment  ...  Segment

TCP
Receive buffer

Application process

Read bytes
TCP Header Format

- Ports plus IP addresses identify a connection
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

SYN, SequenceNum = y

Passive participant (server)

+data

What’s wrong here?
Two-way handshake?

Active participant (client)  Passive participant (server)

Old SYN, SequenceNum = x
New SYN, SequenceNum = q
SYN, SequenceNum = y

Delayed old SYN
Rejected

+data
Three-Way Handshake

- Opens both directions for transfer

Active participant (client)  
Passive participant (server)

- SYN, SequenceNum = x
- SYN + ACK, SequenceNum = y, 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

- How to choose ISNs?
  - Maximize period between reuse
  - Minimize ability to guess (why?)
TCP State Transitions

- **CLOSED**
  - Passive open
  - Close

- **LISTEN**
  - Send SYN
  - SYN/SYN + ACK
  - SYN + ACK/ACK

- **SYN_RCVD**
  - ACK
  - Close /FIN

- **SYN_SENT**
  - SYN/SYN + ACK
  - Send SYN

- **ESTABLISHED**
  - FIN/ACK
  - Close /FIN

- **FIN_WAIT_1**
  - ACK
  - FIN/ACK

- **FIN_WAIT_2**
  - FIN/ACK

- **CLOSING**
  - ACK

- **TIME_WAIT**
  - FIN/ACK
  - Timeout after two segment lifetimes

- **CLOSE_WAIT**
  - FIN/ACK

- **LAST_ACK**
  - ACK

- **CLOSED**
  - ACK
Again, with States

Active participant (client)

- SYN_SENT
- ESTABLISHED

Passive participant (server)

- LISTEN
- SYN_RCVD
- ESTABLISHED

- 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

Web server

FIN_WAIT_1
FIN_WAIT_2
TIME_WAIT
CLOSED

Web browser

FIN
ACK
FIN
ACK
CLOSED_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

- Real life: Abortive close
  - Don’t wait for 2*MSL, simply send Reset packet (RST)
  - Why? Frees up resources on heavily loaded servers
Flow Control

- Sender must transmit data no faster than it can be consumed by the receiver
  - Receiver might be a slow machine
  - App might consume data slowly

- Implement by adjusting the size of the sliding window used at the sender based on receiver feedback about available buffer space
  - This is the purpose of the Advertised Window field
TCP Header Format

- Advertised window is used for flow control
Sender and Receiver Buffering

Sending application

LastByteWritten

LastByteAcked

LastByteSent

Receiving application

LastByteRead

NextByteExpected

LastByteRcvd

■ = available buffer

= buffer in use
Example – Exchange of Packets

Stall due to flow control here

Receiver has buffer of size 4 and application doesn’t read

T=1
SEQ=1
ACK=2; WIN=3
T=2
SEQ=2
ACK=3; WIN=2
T=3
SEQ=3
T=4
SEQ=4
T=5
ACK=4; WIN=1
T=6
ACK=5; WIN=0

April 16, 2002
Example – Buffer at Sender

\[
\begin{array}{cccc}
T=1 & 1 & 2 & 3 & 4 \\
T=2 & 1 & 2 & 3 & 4 \\
T=3 & 1 & 2 & 3 & 4 \\
T=4 & 1 & 2 & 3 & 4 \\
T=5 & 1 & 2 & 3 & 4 \\
T=6 & 1 & 2 & 3 & 4 \\
\end{array}
\]

- =acked
- =sent
- =advertised
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)

<table>
<thead>
<tr>
<th></th>
<th>SrcPort</th>
<th>DstPort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksum</td>
<td></td>
<td>Length</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

April 16, 2002
UDP Delivery

Packets arrive

DeMux

Application process

Application process

Application process

Kemel boundary

Ports

Message Queues
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.
Applications for UDP

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

- Problems from Peterson & Davies:
  - 1.7, 1.13 (a-d), 2.21, 2.23 (a), 4.5, 5.6
  - Extra Credit: 5.17

- Next class: Congestion Control

- No new reading, make sure you’re caught up