Connections

• Focus
  – How do we connect processes?
  – This is the transport layer

• Topics
  – Naming processes
  – Connection setup / teardown
  – Flow control
The Transport Layer

- Builds on the services of the Network Layer
- "TCP/IP"
- Naming/Addressing
- Communication between processes running on hosts
- Stronger guarantees of message delivery make sense
- "TCP/IP"
- This is the first layer that is talking "end-to-end"
Internet Transport Protocols

- **UDP**
  - Datagram abstraction between processes
  - With error detection

- **TCP**
  - Bytestream abstraction between processes
  - With reliability
  - Plus congestion control (later!)
Comparison of TCP/UDP/IP properties

<table>
<thead>
<tr>
<th>TCP</th>
<th>UDP</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection-oriented</td>
<td>Datagram oriented</td>
<td>Datagram oriented</td>
</tr>
<tr>
<td>Reliable byte-stream</td>
<td>Lost packets</td>
<td>Lost packets</td>
</tr>
<tr>
<td>In-order delivery</td>
<td>Reordered packets</td>
<td>Reordered packets</td>
</tr>
<tr>
<td>Single delivery</td>
<td>Duplicate packets</td>
<td>Duplicate packets</td>
</tr>
<tr>
<td>Arbitrarily length</td>
<td>Limited size packets</td>
<td>Limited size packets</td>
</tr>
<tr>
<td>Synchronization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Congestion control</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)
Processes as Endpoints

app stuff
write(), sendto(), send()

OS stuff
Socket file descriptor

Protocol stuff
TCP data
Local Network
Router
Internet
Local Network
Router

read(), recvfrom(), recv()
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
UDP Delivery

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

Application process

TCP
Send buffer

Write bytes

TCP
Receive buffer

Read bytes

Transmit segments
Segment  Segment  ...  Segment
TCP Header Format

- Ports plus IP addresses identify a connection
## TCP Header Format

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td>0-9</td>
</tr>
<tr>
<td>DstPort</td>
<td>10-16</td>
</tr>
<tr>
<td>SequenceNum</td>
<td>16-24</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>25-26</td>
</tr>
<tr>
<td>advertisedWindow</td>
<td>26-30</td>
</tr>
<tr>
<td>Checksum</td>
<td>31-31</td>
</tr>
<tr>
<td>UrgPtr</td>
<td>32-32</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
TCP Header Format

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>4</th>
<th>10</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>SrcPort</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DstPort</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SequenceNum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdvertisedWindow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UrgPtr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TCP Header Format

- Advertised window is used for flow control
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

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

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

PASSIVE OPEN

CLOSED

LISTEN

SYN_RCVD

SYN_SENT

SYN_RCVD

SYN_RCVD

ESTABLISHED

FIN_WAIT_1

FIN_WAIT_2

CLOSING

TIME_WAIT

CLOSE_WAIT

LAST_ACK

CLOSED

CLOSED

CLOSED
Again, with States

Active participant (client)

SYN_SENT

ESTABLISHED

Passive participant (server)

LISTEN

SYN_RCVD

ESTABLISHED

SYN, SequenceNum = x

SYN + ACK, SequenceNum = y,

ACK, Acknowledgment = x + 1

+data

Acknowledgment = y + 1
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 browser

Web server

FIN_WAIT_1

FIN_WAIT_2

TIME_WAIT

CLOSE_WAIT

LAST_ACK

FIN

ACK

ACK

FIN

CLOSE

FIN_WAIT_2
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
Berkeley Sockets interface

- Networking protocols implemented in OS
  - OS must expose a programming API to applications
  - most OSs use the “socket” interface
  - originally provided by BSD 4.1c in ~1982.

- Principle abstraction is a “socket”
  - a point at which an application attaches to the network
  - defines operations for creating connections, attaching to network, sending and receiving data, closing connections
TCP (connection-oriented)

Server
- Socket()
- Bind()
- Listen()
- Accept()
- Block until connect
- Process request
-Recv()
- Send()

Client
- Socket()
- Connect()
- Send()
- Data (request)
- Data (reply)
- Recv()
UDP (connectionless)

```
Server
  Socket()
  Bind()
  Recvfrom()

Block until Data from client
Process request
  Sendto()

Client
  Socket()
  Bind()
  Sendto()
  Recvfrom()
```

Data (request)
Data (reply)
Diversion: TCP SYN cookies

- Solution: SYN cookies
  - Server keeps no state in response to SYN; instead makes client store state
  - Server picks return seq # $y = \oplus$ that encrypts $x$
  - Gets $\oplus + 1$ from sender; unpacks to yield $x$

- Can data arrive before ACK?
Diversion: HTTP on TCP

- How do we reduce the # of messages?
- Delayed ack: wait for 200ms for reply or another pkt arrival
- TCP RST from web server