Transport Layer (TCP/UDP)
Recall the protocol stack

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Programs that use network service</td>
</tr>
<tr>
<td>Transport</td>
<td>Provides end-to-end data delivery</td>
</tr>
<tr>
<td>Network</td>
<td>Send packets over multiple networks</td>
</tr>
<tr>
<td>Link</td>
<td>Send frames over one or more links</td>
</tr>
<tr>
<td>Physical</td>
<td>Send bits using signals</td>
</tr>
</tbody>
</table>
Transport layer

Provides end-to-end connectivity to applications
Transport layer protocols

• 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><strong>Messages</strong></td>
<td>Datagrams (UDP)</td>
<td>Streams (TCP)</td>
</tr>
<tr>
<td><strong>Bytestream</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison of Internet transports

• TCP is full-featured, UDP is a glorified packet

<table>
<thead>
<tr>
<th>TCP (Streams)</th>
<th>UDP (Datagrams)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>Datagrams</td>
</tr>
<tr>
<td>Bytes are delivered once, reliably, and in order</td>
<td>Messages may be lost, reordered, duplicated</td>
</tr>
<tr>
<td>Arbitrary length content</td>
<td>Limited message size</td>
</tr>
<tr>
<td>Flow control matches sender to receiver</td>
<td>Can send regardless of receiver state</td>
</tr>
<tr>
<td>Congestion control matches sender to network</td>
<td>Can send regardless of network state</td>
</tr>
</tbody>
</table>
Socket API

• Simple 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 (Streams and Datagrams)
Socket API (2)

- **Sockets** let apps attach to the local network at different ports
Socket API (3)

• 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
Ports

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

• Servers often bind to “well-known ports”
  • <1024, require administrative privileges

• Clients often assigned “ephemeral” ports
  • Chosen by OS, used temporarily
### Some Well-Known Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/20, 21</td>
<td>FTP</td>
<td>File transfer</td>
</tr>
<tr>
<td>TCP/22</td>
<td>SSH</td>
<td>Remote login, replacement for Telnet</td>
</tr>
<tr>
<td>TCP/25</td>
<td>SMTP</td>
<td>Email</td>
</tr>
<tr>
<td>TCP/80</td>
<td>HTTP</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>TCP/443</td>
<td>HTTPS</td>
<td>Secure Web (HTTP over SSL/TLS)</td>
</tr>
<tr>
<td>TCP/3306</td>
<td>MYSQL</td>
<td>MYSQL database access</td>
</tr>
<tr>
<td>UDP/53</td>
<td>DNS</td>
<td>Domain name service</td>
</tr>
</tbody>
</table>

Full list: [https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt](https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt)
Topics

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

• Used by apps that don’t want reliability or bytestreams
  • Like what?
User Datagram Protocol (UDP)

• Used by apps that don’t want reliability or bytestreams
  • Voice-over-IP
  • DNS
  • DHCP
  • Games

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

Client (host 1)        Time        Server (host 2)

↓

request

↓

reply
Datagram Sockets (2)

Client (host 1)       Time       Server (host 2)
1: socket
4: sendto
5: recvfrom*
7: close

1: socket
2: bind
3: recvfrom*

request
reply

6: sendto
7: close

*= call blocks
UDP Buffering

Application

Ports

Transport (UDP)

Message queues

Port Mux/Demux

Network (IP)

packet
UDP Header

- Uses ports to identify sending and receiving application processes
- Datagram length up to 64K
- Checksum (16 bits) for reliability
**UDP Header (2)**

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

<table>
<thead>
<tr>
<th>Source address</th>
<th>Protocol = 17</th>
<th>UDP length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TCP
TCP

• TCP Consists of 3 primary phases:
  • Connection Establishment (Setup)
  • Sliding Windows/Flow Control
  • Connection Release (Teardown)
Connection Establishment

• Both sender and receiver must be ready before we start the transfer of data
  • 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
    • Like “dialing” for a telephone call
Three-Way Handshake

- Used in TCP; opens connection for data in both directions
- Each side probes the other with a fresh Initial Sequence Number (ISN)
  - Sends on a SYNchronize segment
  - Echo on an ACKnowledge segment
- Chosen to be robust even against delayed duplicates
Three-Way Handshake (2)

• Three steps:
  • Client sends SYN(x)
  • Server replies with SYN(y)ACK(x+1)
  • Client replies with ACK(y+1)
  • SYNs are retransmitted if lost

• Sequence and ack numbers carried on further segments
Three-Way Handshake (3)

• Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  • Improbable, but anyhow ...

Active party (client)

Passive party (server)

SYN (SEQ=x)

(SEQ=x+1, ACK=z+1)
Three-Way Handshake (4)

- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow ...

- Connection will be cleanly rejected on both sides 😊

[Diagram showing the three-way handshake with active and passive parties, including SYN and ACK messages with sequence and acknowledgment numbers.]
TCP Connection State Machine

- Captures the states ([ ]) and transitions (-->)
  - A/B means event A triggers the transition, with action B

Both parties run instances of this state machine
TCP Connections (2)

• Follow the path of the client:
TCP Connections (3)

- And the path of the server:
TCP Connections (4)

• Again, with states ...

Active party (client) Passive party (server)

CLOSED SYN_SENT

SYN (SEQ=x) SYN (SEQ=y, ACK=x+1)

SYN (SEQ=x+1, ACK=y+1)

Time

CLOSED LISTEN SYN_RCVD ESTABLISHED

ESTABLISHED
TCP Connections (5)

• Finite state machines are a useful tool to specify and check the handling of all cases that may occur

• TCP allows for simultaneous open
  • i.e., both sides open instead of the client-server pattern
  • Try at home to confirm it works 😊