Computer Chat

- How do we make computers talk?

- How are they interconnected?

Internet Protocol (IP)
Internet Protocol (IP)

- Datagram (packet) protocol
- Best-effort service
  - Loss
  - Reordering
  - Duplication
  - Delay
- Host-to-host delivery

IP Address

- 32-bit identifier
- Dotted-quad: 192.118.56.25
- www.mkp.com -> 167.208.101.28
- Identifies a host interface (not a host)
Transport Protocols

Best-effort not sufficient!

- Add services on top of IP
- User Datagram Protocol (UDP)
  - Data checksum
  - Best-effort
- Transmission Control Protocol (TCP)
  - Data checksum
  - Reliable byte-stream delivery
  - Flow and congestion control

Ports

Identifying the ultimate destination

- IP addresses identify hosts
- Host has many applications
- Ports (16-bit identifier)

<table>
<thead>
<tr>
<th>Application</th>
<th>WWW</th>
<th>E-mail</th>
<th>Telnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>80</td>
<td>25</td>
<td>23</td>
</tr>
</tbody>
</table>

192.18.22.13
**Sockets**

- Identified by protocol and local/remote address/port
- Applications may refer to many sockets

![Diagram of sockets and protocol]

**Clients and Servers**

- **Client:** Initiates the connection

  **Client:** Bob  
  “Hi. I’m Bob.”

  **Server:** Jane  
  “Hi, Bob. I’m Jane”

  “Nice to meet you, Jane.”

- **Server:** Passively waits to respond
TCP Client/Server Interaction

Server starts by getting ready to receive client connections...

Client
- Create a TCP socket
- Communicate
- Close the connection

Server
- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection

TCP Client/Server Interaction

ServerSocket servSock = new ServerSocket(servPort);

Client
- Create a TCP socket
- Communicate
- Close the connection

Server
- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection
TCP Client/Server Interaction

Server is now blocked waiting for connection from a client
TCP Client/Server Interaction

Later, a client decides to talk to the server...

Client
- Create a TCP socket
- Communicate
- Close the connection

Server
- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection

Socket socket = new Socket(server, servPort);

Client
- Create a TCP socket
- Communicate
- Close the connection

Server
- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection
TCP Client/Server Interaction

OutputStream out = socket.getOutputStream();
out.write(byteBuffer);

Client
Create a TCP socket
Communicate
Close the connection

Server
Create a TCP socket
Repeatedly:
Accept new connection
Communicate
Close the connection

TCP Client/Server Interaction

Socket clntSock = servSock.accept();

Client
Create a TCP socket
Communicate
Close the connection

Server
Create a TCP socket
Repeatedly:
Accept new connection
Communicate
Close the connection
TCP Client/Server Interaction

Client
- Create a TCP socket
- Communicate
- Close the connection

Server
- Create a TCP socket
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection

TCP Client/Server Interaction

close(sock);

Client
- Create a TCP socket
- Establish connection
- Communicate
- Close the connection

Server
- Create a TCP socket
- Bind socket to a port
- Set socket to listen
- Repeatedly:
  - Accept new connection
  - Communicate
  - Close the connection
## TCP Tidbits

- Client knows server address and port
- No correlation between `send()` and `recv()`

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out.write(&quot;Hello Bob&quot;)</code></td>
<td><code>in.read() -&gt; &quot;Hello &quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>in.read() -&gt; &quot;Bob&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>out.write(&quot;Hi &quot;)</code></td>
</tr>
<tr>
<td></td>
<td><code>out.write(&quot;Jane&quot;)</code></td>
</tr>
<tr>
<td></td>
<td><code>in.read() -&gt; &quot;Hi Jane&quot;</code></td>
</tr>
</tbody>
</table>

## Closing a Connection

- `close()` used to delimit communication
- Analogous to EOF

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out.write(string)</code></td>
<td><code>in.read(buffer)</code></td>
</tr>
<tr>
<td>while (not received entire string)</td>
<td>while (client has not closed connection)</td>
</tr>
<tr>
<td><code>in.read(buffer)</code></td>
<td><code>out.write(buffer)</code></td>
</tr>
<tr>
<td><code>out.write(buffer)</code></td>
<td><code>in.read(buffer)</code></td>
</tr>
<tr>
<td><code>close(socket)</code></td>
<td><code>close(client socket)</code></td>
</tr>
</tbody>
</table>
Constructing Messages

...beyond simple strings

TCP/IP Byte Transport

- TCP/IP protocols transports *bytes*

- Application protocol provides semantics

Here are some bytes. I don’t know what they mean.

I’ll pass these to the app. It knows what to do.
Application Protocol

- Encode information in bytes
- Sender and receiver must agree on semantics
- Data encoding
  - Primitive types: strings, integers, and etc.
  - Composed types: message with fields

Primitive Types

- String
  - Character encoding: ASCII, Unicode, UTF
  - Delimit: length vs. termination character

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>77</th>
<th></th>
<th>0</th>
<th>111</th>
<th></th>
<th>0</th>
<th>109</th>
<th></th>
<th>0</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M</td>
<td>o</td>
<td>m</td>
<td>\n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>77</td>
<td>111</td>
<td>109</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Primitive Types**

- **Integer**
  - Strings of character encoded decimal digits
  - Advantage: 1. Human readable
    2. Arbitrary size
  - Disadvantage: 1. Inefficient
    2. Arithmetic manipulation

<table>
<thead>
<tr>
<th>49</th>
<th>55</th>
<th>57</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>48</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>'1'</td>
<td>'7'</td>
<td>'9'</td>
<td>'9'</td>
<td>'8'</td>
<td>'7'</td>
<td>'0'</td>
<td>'n'</td>
</tr>
</tbody>
</table>

**Network byte order (Big-Endian)**
- Use for multi-byte, binary data exchange
- htonl(), htons(), ntohl(), ntohs()
Message Composition

- Message composed of fields
  - Fixed-length fields
    
    | integer | short | short |
    |---------|-------|-------|

  - Variable-length fields
    
    | M | i | k | e | 1 | 2 | \n    |---|---|---|---|---|---|