Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
- 2.5 DNS

- 2.6 P2P applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - it relies on transport infrastructure on other side of door which brings message to socket at receiving process

9/26/2014
Socket programming

Goal: learn how to write client and server programs which communicate by sending data into sockets, reading data out of sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- sockets are explicitly created, used, then released by applications
- client/server paradigm
- API: choice of a transport protocol and ability to specify a few parameters
  - reliable byte stream – TCP
  - unreliable datagram – UDP
Client-server architecture

server:
- always-on host
- permanent IP address
- data centers for scaling

clients:
- communicate with server (speak first)
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
## User Datagram Protocol (UDP): Datagram Socket

<table>
<thead>
<tr>
<th>UDP</th>
<th>Postal Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>single socket to receive messages from all senders</td>
<td>single mailbox to receive letters from all senders</td>
</tr>
<tr>
<td>no guarantee of delivery</td>
<td>unreliable</td>
</tr>
<tr>
<td>not necessarily in-order delivery</td>
<td>not necessarily in-order delivery</td>
</tr>
<tr>
<td>datagrams sent independently</td>
<td>letters sent independently</td>
</tr>
<tr>
<td>must address each packet</td>
<td>must address each mail</td>
</tr>
</tbody>
</table>

### Example UDP applications:
- DNS, SNMP
Connectionless demultiplexing

- UDP socket identified by two-tuple:
  (dest IP address, dest port number)

- When host receives UDP segment:
  - directs UDP segment to socket with destination port number

- IP datagrams from different sources directed to same UDP socket
Transmission Control Protocol (TCP): Stream Socket

- reliable - guaranteed delivery
- byte stream - in-order delivery
- connection-oriented - one connection for each client
- connection setup followed by data transfer

Example TCP applications
Web, SMTP, Telnet
Connection-oriented demux

- Server has welcome and connection sockets
  - welcome socket is identified by server’s IP address and a port number
- TCP connection socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number

- Server may support many simultaneous TCP connection sockets with clients:
  - each connection socket and the welcome socket have the same port number in server host
  - receiving host uses all four values to direct segment to appropriate connection socket
Knowing What Port Number To Use

- **Well-known ports for servers of popular applications**
  - e.g., port 80 for Web and port 25 for e-mail
  - see http://www.iana.org/assignments/port-numbers

- **Ephemeral ports for servers of other applications**
  - server programmer picks an unused ephemeral (i.e., temporary) port
    - between 1024 and 65535
Socket programming with TCP

Before client contact:
- server process must first be running
- server must have created socket (door) that welcomes client’s contact

Client contacts server by:
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client makes connect call: client TCP establishes connection to server TCP

When contacted by client, server TCP creates a new connection socket for server process to communicate with client
- allows server to talk with multiple clients
- source port numbers used to distinguish clients (more in Chap 3)

TCP provides reliable, in-order transfer of byte stream between client and server
Socket API for TCP (BSD Unix)

**Client side**
- `socket()`, returns client socket id
- `connect()`, need to specify server IP address and port, sends conn req
- `send()`, sends to client socket
- `recv()`, receives from client socket
- `close()`, closes connection

**Server side**
- `socket()`, returns server socket id
- `bind()`, binds server socket to server IP address and port
- `listen()`, willing to accept conn req on server socket
- `accept()`, accepts new conn req and returns its connection socket id
- `recv()`, receives from connection socket
- `send()`, sends to connection socket
- `close()`, closes connection

*note: OS supplies local IP address and port for client*

9/26/2014
Client/server socket interaction: TCP

**server** (running on **hostid**)  

- create socket, port=x, for incoming request:
  ```python
  serverSocket = socket()
  ```
- wait for incoming connection request:
  ```python
  connectionSocket = serverSocket.accept()
  ```
- read request from `connectionSocket`
- write reply to `connectionSocket`
- close `connectionSocket`

**client**

- create socket:
  ```python
  clientSocket = socket()
  ```
- connect to **hostid**, port=x
- send request using `clientSocket`
- read reply from `clientSocket`
- close `clientSocket`

TCP connection setup

Application Layer

9/26/2014
Example: TCP client in Python

from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence: ')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
Example: TCP server in Python

```python
from socket import *
s
serverPort = 12000

serverSocket = socket(AF_INET,
                      SOCK_STREAM)

serverSocket.bind(("",serverPort))

serverSocket.listen(1)

print 'The server is ready to receive'

while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)

    connectionSocket.close()  # close connection to this client (but not welcoming socket)
```

- Create TCP welcoming socket
- Server begins listening for incoming TCP requests
- Server waits on accept() for incoming requests, new socket created on return
- Read bytes from connection socket
- Close connection to this client (but not welcoming socket)
Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
- 2.5 DNS

- 2.6 P2P applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
Socket programming with UDP

UDP: no “connection” between client and server
- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server
Socket API for UDP (BSD Unix)

- **Client side**
  - `socket()`, returns client socket id
  - `sendto()`, sends to client socket, need to specify destination's IP address and port
  - `recvfrom()`, receives from client socket: data and sender's IP address and port
  - `bind()`, optional

- **Server side**
  - `socket()`, returns server socket id
  - `bind()`, binds server socket to server IP address and port
  - `recvfrom()`, receives from server socket: data and sender's IP address and port
  - `sendto()`, sends to server socket, need to specify destination's IP address and port

- note: needs timeout management;
  OS supplies local IP address and port for client if `bind()` not used

9/26/2014
Client/server socket interaction: UDP

server (running on serverIP)

create socket, port= x:
serverSocket = socket(AF_INET, SOCK_DGRAM)

read datagram from serverSocket

write reply to serverSocket specifying client address, port number

client

create socket:
clientSocket = socket(AF_INET, SOCK_DGRAM)

create datagram with server IP address and port=x; send datagram via clientSocket

read datagram from clientSocket

write reply to serverSocket

close clientSocket

9/26/2014
Example: UDP client in Python

```python
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence: ')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

create UDP socket for server

Attach server name, port to message; send into socket

read reply characters from socket into string

Application Layer
Example: UDP server in Python

```python
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,
                      SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

**Application Layer**
The end