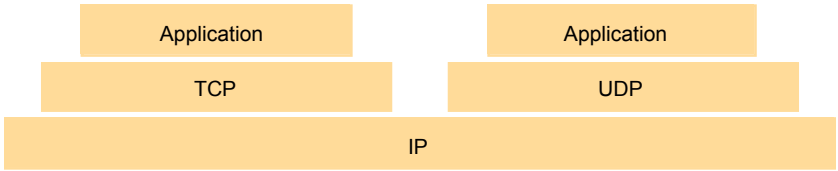
Network Programming using sockets

The programmer's conceptual view of a TCP/IP Internet



Internet Connections

Clients and servers communicate by sending streams of bytes over *connections*:

- Point-to-point, full-duplex (2-way communication), and reliable.

A socket is an endpoint of a connection

- Socket address is an `IPaddress:port` pair

A *port* is a 16-bit integer that identifies a process:

- *Ephemeral port*: Assigned automatically on client when client makes a connection request
- Well-known port: Associated with some service provided by a server (e.g., port 80 is associated with Web servers)

A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)

- `(cliaddr:cliport, servaddr:servport)`

3

Clients

Examples of client programs

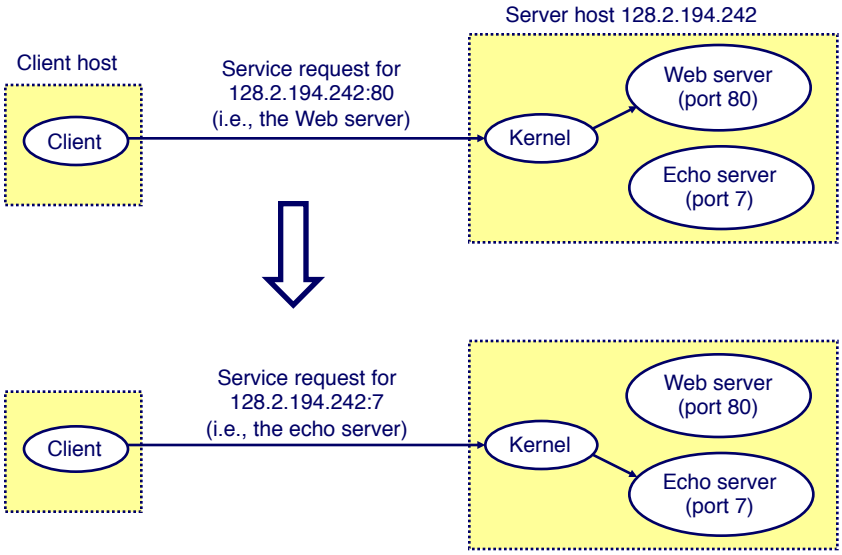
- Web browsers, `ftp`, `telnet`, `ssh`

How does a client find the server?

- The IP address in the server socket address identifies the host *(more precisely, an adapter on the host)*
- The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
- Examples of well known ports
  - Port 7: Echo server
  - Port 23: Telnet server
  - Port 25: Mail server
  - Port 80: Web server

4

Using Ports to Identify Services



Servers

Servers are long-running processes (daemons).

- **Created at boot-time (typically) by the init process (process 1)**
- **Run continuously until the machine is turned off.**

Each server waits for requests to arrive on a well-known port associated with a particular service.

- **Port 7: echo server**
- **Port 23: telnet server**
- **Port 25: mail server**
- **Port 80: HTTP server**

A machine that runs a server process is also often referred to as a "server."

Server Examples

Web server (port 80)

- Resource: files/compute cycles (CGI programs)
- Service: retrieves files and runs CGI programs on behalf of the client

FTP server (20, 21)

- Resource: files
- Service: stores and retrieve files

See `/etc/services` for a comprehensive list of the services available on a Linux machine.

Telnet server (23)

- Resource: terminal
- Service: proxies a terminal on the server machine

Mail server (25)

- Resource: email "spool" file
- Service: stores mail messages in spool file

7

Sockets Interface

Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols.

Provides a user-level interface to the network.

Underlying basis for all Internet applications.

Based on client/server programming model.

8

Sockets

What is a socket?

- **To the kernel, a socket is an endpoint of communication.**
- **To an application, a socket is a file descriptor that lets the application read/write from/to the network.**
  - **All Unix I/O devices, including networks, are modeled as files.**

Clients and servers communicate with each other by reading from and writing to socket descriptors.

The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors.

9

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

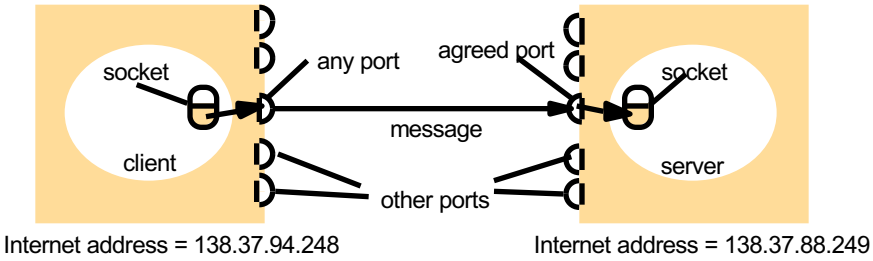
- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket

a *host-local*, *application-created/owned*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another (remote or local) application process

10

Sockets and ports

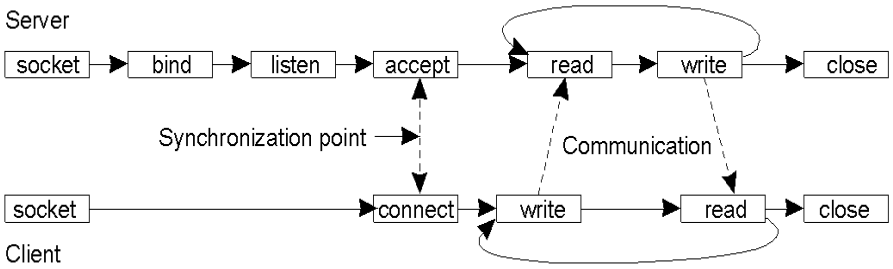


Berkeley Sockets (1)

Socket primitives for TCP/IP.

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Berkeley Sockets (2)



Connection-oriented communication pattern using sockets.

13

Sockets used for streams

Requesting a connection

```
s = socket(AF_INET, SOCK_STREAM,0)
●
●
connect(s, ServerAddress)
●
●
write(s, "message", length)
```

Listening and accepting a connection

```
s = socket(AF_INET, SOCK_STREAM,0)
●
bind(s, ServerAddress);
listen(s,5);
●
sNew = accept(s, ClientAddress);
●
n = read(sNew, buffer, amount)
```

ServerAddress and *ClientAddress* are socket addresses

14

Socket programming with TCP

Client must contact server

- 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 creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

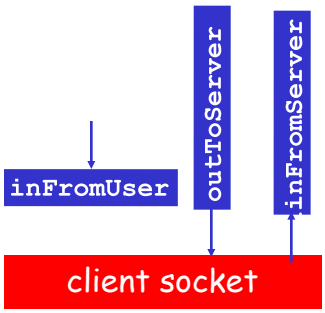
Socket programming with TCP

Example client-server app:

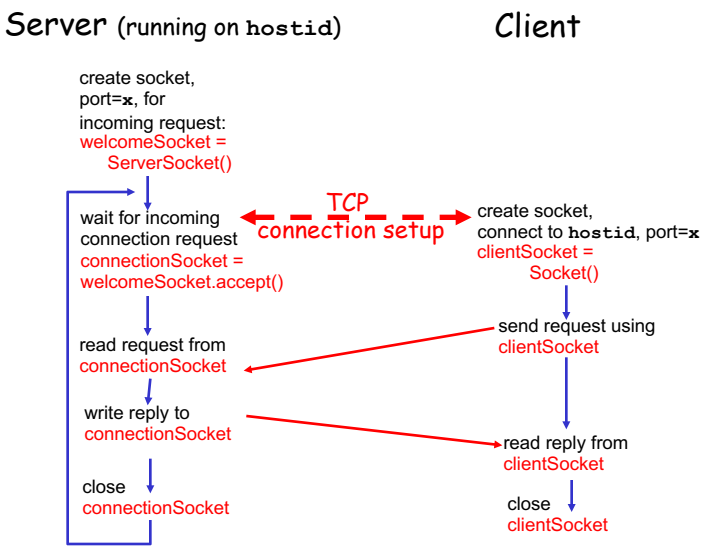
- client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)

Input stream: sequence of bytes into process

Output stream: sequence of bytes out of process



Client/server socket interaction: TCP



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

  }
}
```


Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
    String clientSentence;
    String capitalizedSentence;
```

Create welcoming socket at port 6789

```
    ServerSocket welcomeSocket = new ServerSocket(6789);

    while(true) {
```

Wait, on welcoming socket for contact by client

```
      Socket connectionSocket = welcomeSocket.accept();
```

Create input stream, attached to socket

```
      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Example: Java server (TCP), cont

Create output stream, attached to socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
    }
  }
}
```

End of while loop, loop back and wait for another client connection

Socket programming with UDP

UDP: no "connection" between client and server

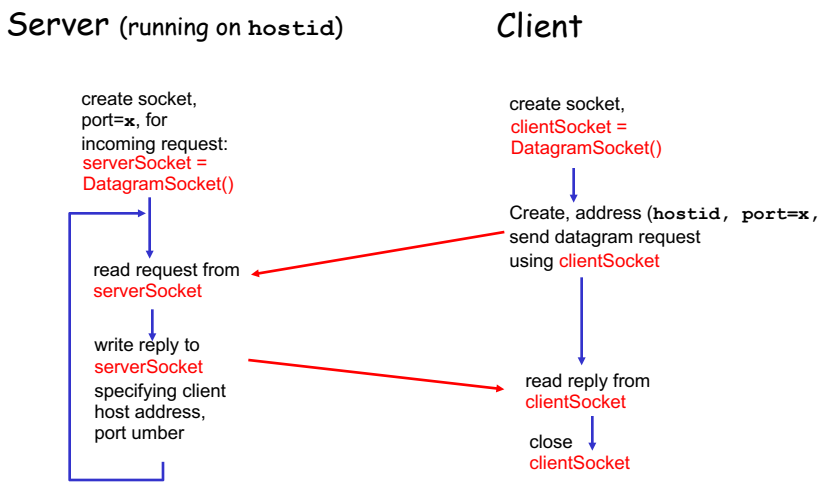
- no handshaking
- sender explicitly attaches IP address and port of destination
- server must extract IP address, port of sender from received datagram

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

Client/server socket interaction: UDP



Sockets used for datagrams

Sending a message

```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
bind(s, ClientAddress)
•
•
sendto(s, "message", ServerAddress)
```

Receiving a message

```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
bind(s, ServerAddress)
•
•
amount = recvfrom(s, buffer, from)
```

ServerAddress and *ClientAddress* are socket addresses

Example: Java client (UDP)

Create input stream

Create client socket

Translate hostname to IP address using DNS

```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
   {
      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

25

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
      DatagramPacket sendPacket =
        new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

      clientSocket.send(sendPacket);

      DatagramPacket receivePacket =
        new DatagramPacket(receiveData, receiveData.length);

      clientSocket.receive(receivePacket);

      String modifiedSentence =
         new String(receivePacket.getData());

      System.out.println("FROM SERVER:" + modifiedSentence);
      clientSocket.close();
      }
}
```

26

Example: Java server (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {
      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {
          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

Example: Java server (UDP), cont

```
          String sentence = new String(receivePacket.getData());
          InetAddress IPAddress = receivePacket.getAddress();
          int port = receivePacket.getPort();

                  String capitalizedSentence = sentence.toUpperCase();

          sendData = capitalizedSentence.getBytes();
          DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress,
                               port);
          serverSocket.send(sendPacket);
        }
    }
}
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram