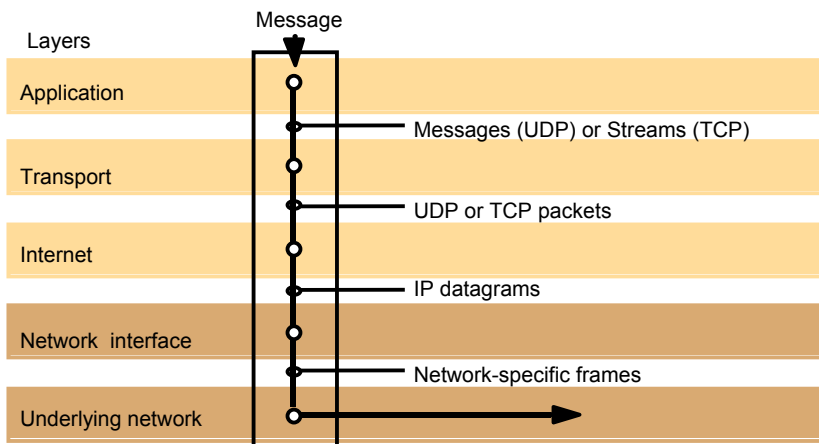


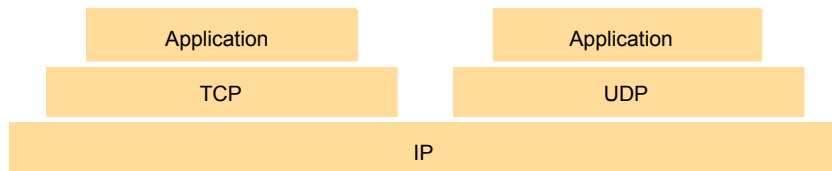
Communication in distributed systems:
network programming using sockets

Operating Systems

TCP/IP layers



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



Socket programming

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

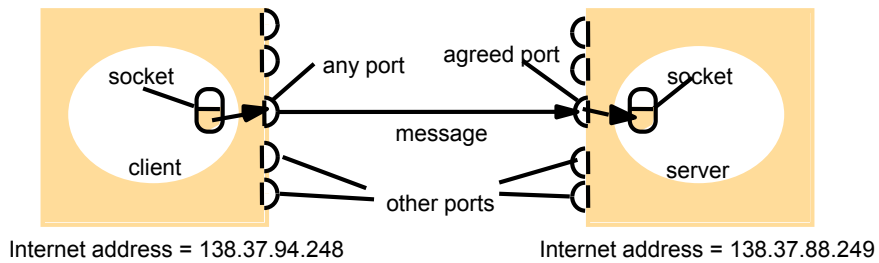
Socket API

- 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

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

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

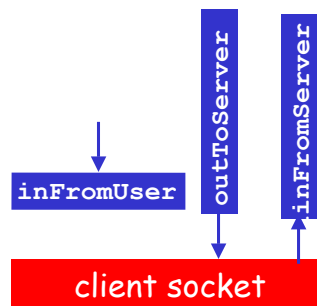
Socket programming with TCP

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

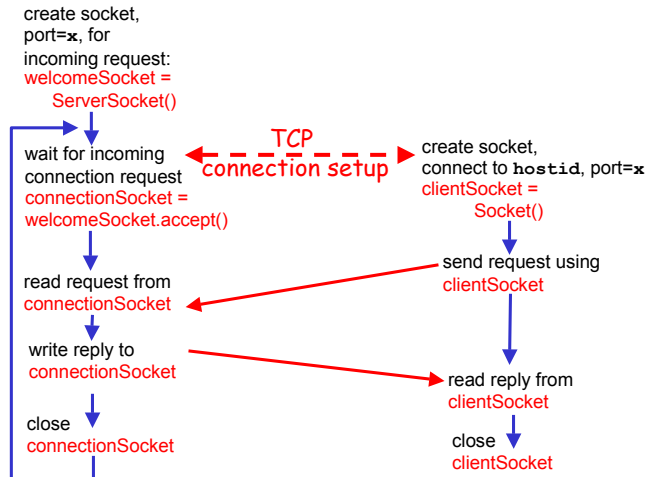
Output stream: sequence of bytes out of process



Client/server socket interaction: TCP

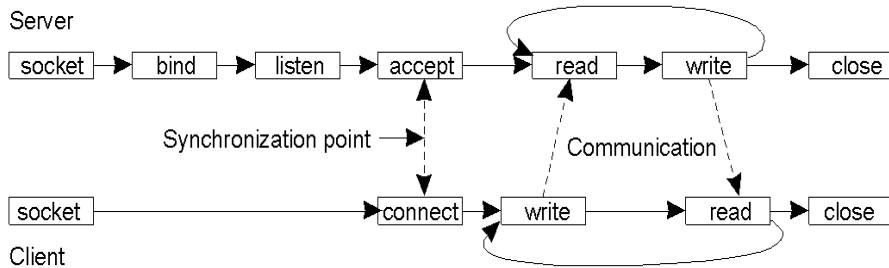
Server (running on `hostid`)

Client



Network Programming 9

Berkeley Sockets (2)



- Connection-oriented communication pattern using sockets.

Network Programming 10

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

Network Programming 11

Example: Java client (TCP)

```
import java.io.*;  
import java.net.*;  
class TCPClient {  
  
    public static void main(String argv[]) throws Exception  
    {  
        String sentence;  
        String modifiedSentence;  
  
        Create input stream → BufferedReader inFromUser =  
                                new BufferedReader(new InputStreamReader(System.in));  
  
        Create client socket, connect to server → Socket clientSocket = new Socket("hostname", 6789);  
  
        Create output stream attached to socket → DataOutputStream outToServer =  
                                                    new DataOutputStream(clientSocket.getOutputStream());
```

Network Programming 12

Example: Java client (TCP), cont.

```
        Create  
input stream ] → BufferedReader inFromServer =  
attached to socket ] new BufferedReader(new  
                        InputStreamReader(clientSocket.getInputStream()));  
  
        Send line ] → sentence = inFromUser.readLine();  
to server ] outToServer.writeBytes(sentence + '\n');  
  
        Read line ] → modifiedSentence = inFromServer.readLine();  
from server ] System.out.println("FROM SERVER: " + modifiedSentence);  
  
        clientSocket.close();  
  
    }  
}
```

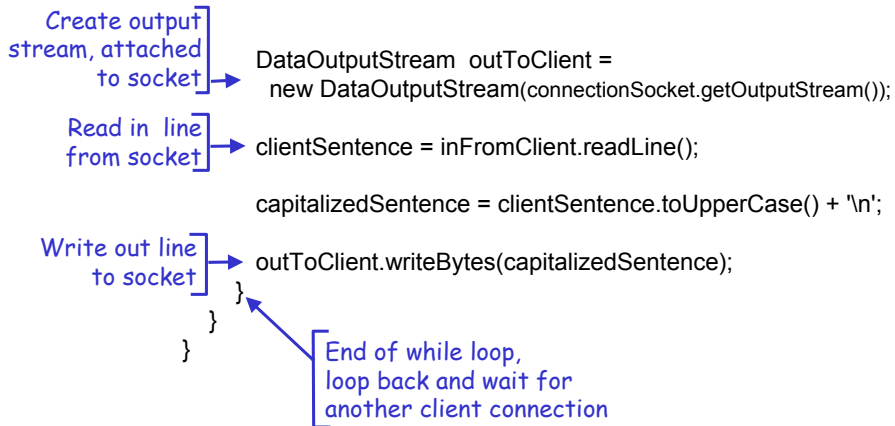
Network Programming 13

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 ] → ServerSocket welcomeSocket = new ServerSocket(6789);  
at port 6789 ]  
  
        Wait, on welcoming ]  
socket for contact ] → Socket connectionSocket = welcomeSocket.accept();  
by client ]  
  
        Create input ]  
stream, attached ] → BufferedReader inFromClient =  
to socket ] new BufferedReader(new  
                InputStreamReader(connectionSocket.getInputStream()));  
  
    }  
}
```

Network Programming 14

Example: Java server (TCP), cont



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

application viewpoint

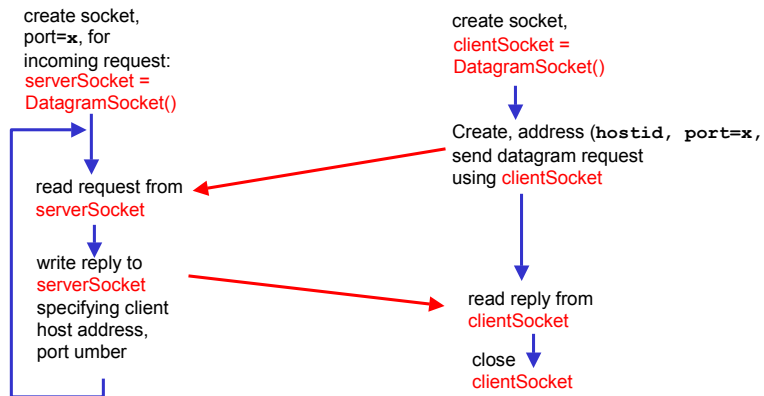
UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

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

Client/server socket interaction: UDP

Server (running on `hostid`)

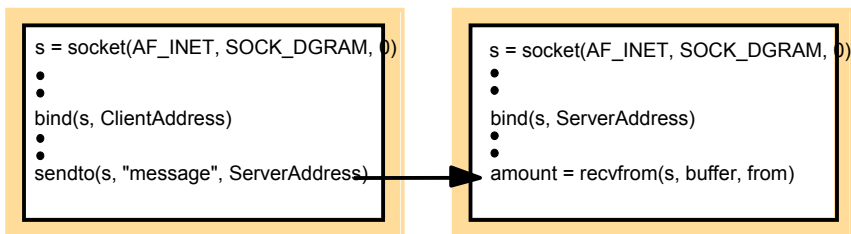
Client



Sockets used for datagrams

Sending a message

Receiving a message



ServerAddress and *ClientAddress* are socket addresses

Example: Java client (UDP)

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

class UDPClient {
    public static void main(String args[]) throws Exception
    {
        Create input stream → BufferedReader inFromUser =
                               new BufferedReader(new InputStreamReader(System.in));
        Create client socket → DatagramSocket clientSocket = new DatagramSocket();
        Translate hostname to IP address using DNS → InetAddress IPAddress = InetAddress.getByName("hostname");

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

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

Network Programming 19

Example: Java client (UDP), cont.

```
Create datagram with data-to-send, length, IP addr, port → DatagramPacket sendPacket =
                                                              new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
Send datagram to server → clientSocket.send(sendPacket);
                                                                    DatagramPacket receivePacket =
                                                                    new DatagramPacket(receiveData, receiveData.length);
Read datagram from server → clientSocket.receive(receivePacket);

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

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

Network Programming 20

Example: Java server (UDP)

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

class UDPServer {
    public static void main(String args[]) throws Exception
    {
        Create datagram socket at port 9876 → DatagramSocket serverSocket = new DatagramSocket(9876);

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

        while(true)
        {
            Create space for received datagram → DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            Receive datagram → serverSocket.receive(receivePacket);
        }
    }
}
```

Network Programming 21

Example: Java server (UDP), cont

```
String sentence = new String(receivePacket.getData());
Get IP addr port #, of sender → InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

Create datagram to send to client → DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress,
    port);

Write out datagram to socket → serverSocket.send(sendPacket);
}
}
}
End of while loop, loop back and wait for another datagram
```

Network Programming 22