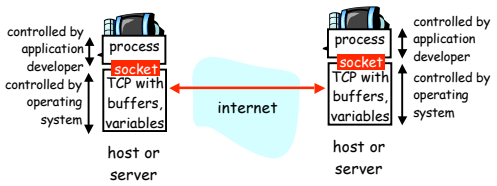
Socket Programming

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of bytes from one process to another



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

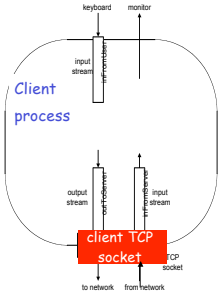
Client contacts server by:
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

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

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

Stream jargon

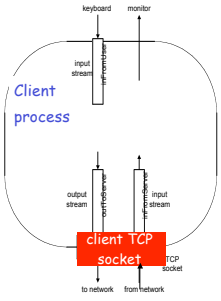
- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, eg, keyboard or socket.
- An output stream is attached to an output source, eg, monitor or socket.



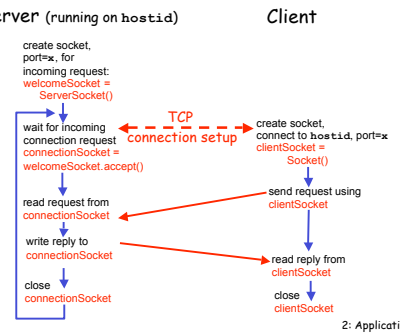
Socket programming with TCP

Example client-server app:

1) client reads line from standard input (`inFromUser` stream) , sends to server via socket (`outToServer` stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (`inFromServer` stream)



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.

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

      sentence = inFromUser.readLine();

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

      modifiedSentence = inFromServer.readLine();

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

      clientSocket.close();

   }
}
```

Create input stream attached to socket

Send line to server

Read line from server

Example: Java server (TCP)

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

class TCPServer {

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

      ServerSocket welcomeSocket = new ServerSocket(6789);

      while(true) {

        Socket connectionSocket = welcomeSocket.accept();

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

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont

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

        clientSentence = inFromClient.readLine();

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

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

Create output stream, attached to socket

Read in line from socket

Write out line to socket

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

Socket programming *with UDP*

UDP: no "connection" between client and server

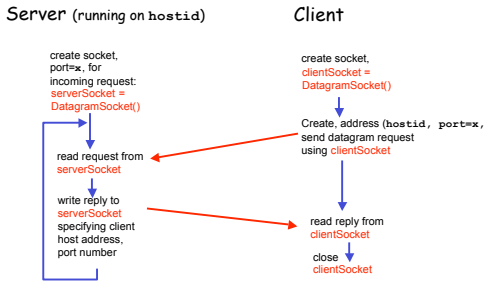
- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

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

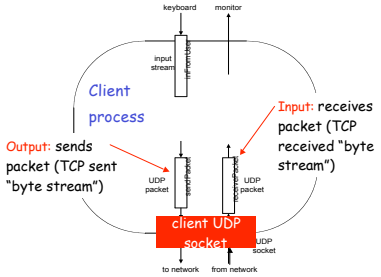
application viewpoint

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

Client/server socket interaction: UDP



Example: Java client (UDP)



Example: Java client (UDP)

```
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();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Example: Java client (UDP), cont.

```
     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();
     }
}
```

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

Send datagram to server

Read datagram from server

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

Building a simple Web server

- handles one HTTP request
- accepts the request
- parses header
- obtains requested file from server's file system
- creates HTTP response message:
  - header lines + file
- sends response to client
- after creating server, you can request file using a browser (eg IE explorer)
- see text for details