

Capa Aplicación: Programación de sockets

ELO322: Redes de Computadores Agustín J. González

Este material está basado en:

- Material de apoyo al texto *Computer Networking: A Top Down Approach Featuring the Internet 3rd* edition. Jim Kurose, Keith Ross Addison-Wesley, 2004.

Capítulo 2: Capa Aplicación

- 2.1 Principios de la aplicaciones de red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo Electrónico
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P Compartición de archivos
- 2.7 Programación de sockets con TCP
- 2.8 Programación de sockets con UDP
- 2.9 Construcción de un servidor WEB

Programación de Sockets

Objetivo: aprender cómo construir aplicaciones cliente servidor que se comunican usando sockets

API para sockets

- Fue introducida en BSD4.1 UNIX, 1981
- El socket es explícitamente creado, usado, y cerrado (o terminado) por las aplicaciones
- Sigue el modelo cliente/servidor
- Hay dos tipos de servicios de transporte vía el API de socket:
 - Datagramas no confiables
 - Orientado a un flujo de bytes, es confiable

sockets

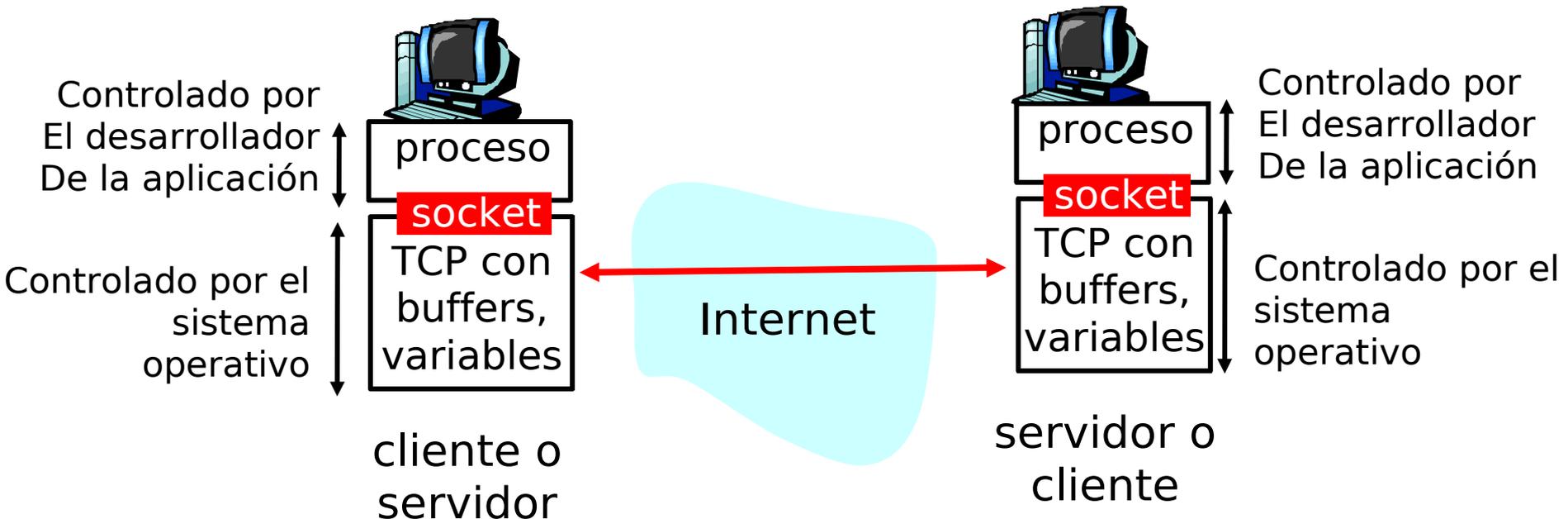
Son *locales al host, creados por la aplicación,*
Es una interfaz controlada por el OS
(una “puerta”) a través de la cual el proceso aplicación puede tanto **enviar como recibir** mensajes a/desde otro proceso remoto de la aplicación

Programación de Sockets con TCP

Transmission Control Protocol

Socket: es una puerta entre el proceso aplicación y el protocolo de transporte de extremo a extremo (UCP o TCP)

Servicio TCP: transferencia confiable de **bytes** desde un proceso a otro



Programación de Sockets con TCP

El cliente debe contactar al servidor

- Proceso servidor debe estar corriendo primero
- Servidor debe tener creado el socket (puerta) que acoge al cliente

El cliente contacta al servidor por:

- La creación de un socket TCP local para el cliente
- Especifica la dirección IP, número de puerto del proceso servidor
- Una vez que el **cliente crea el socket**: el socket establece una conexión TCP al servidor

- Cuando el servidor es contactado por el cliente, el **servidor TCP crea otro socket** para que el proceso servidor se comuniquen con ese cliente
 - Permite que un servidor hable con múltiples clientes
 - IP y Número de puerto fuente distingue a cada cliente (**más adelante más sobre esto**)

Punto de vista de la aplicación
TCP provee transferencias de bytes confiables y en orden. Es un pipeline (o "tubería") de datos entre el cliente y servidor

Sockets creados en relación cliente/servidor usando TCP

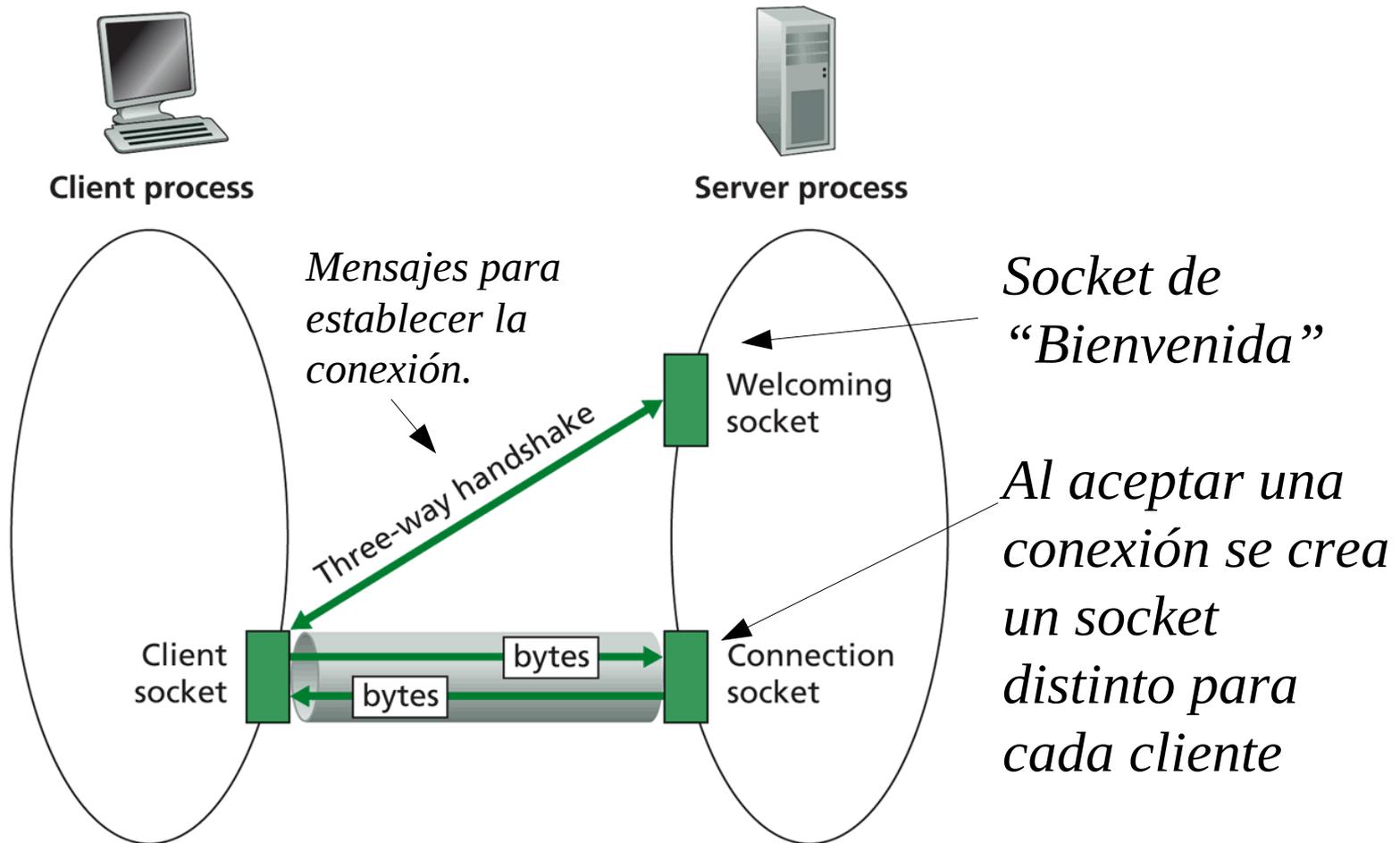


Figure 2.27 ♦ Client socket, welcoming socket, and connection socket, aplicación

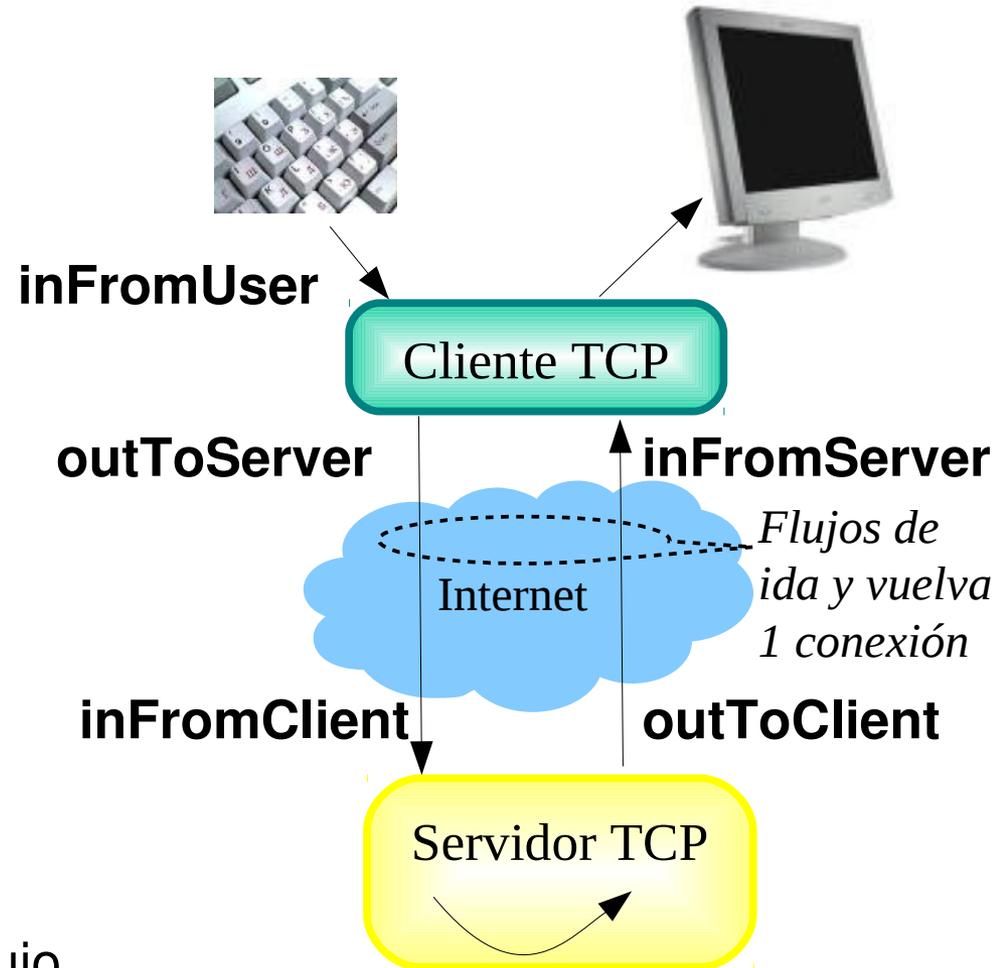
Jerga de flujos (Stream)

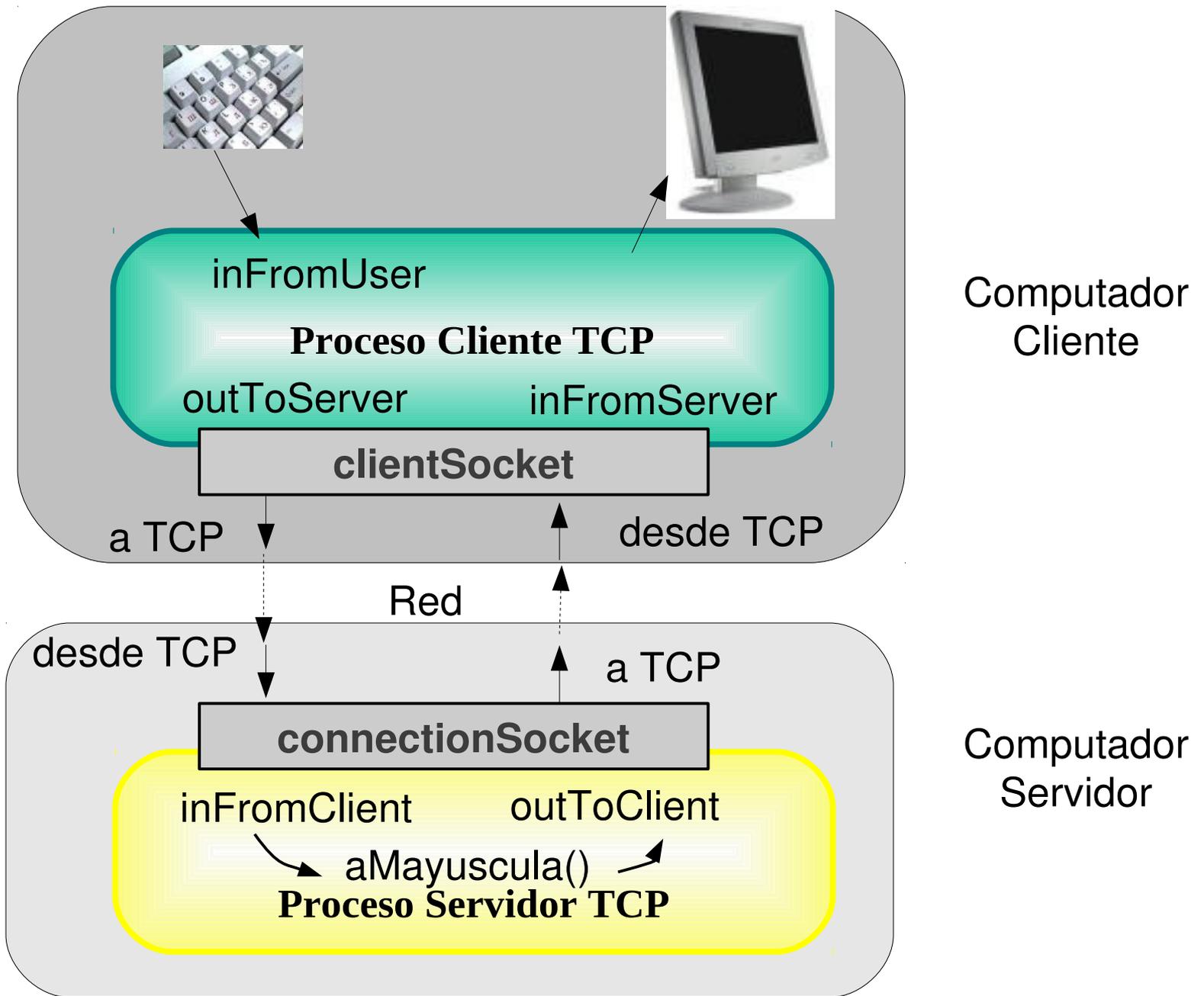
- ▣ Un **stream (flujo)** es una secuencia de bytes que fluyen hacia o desde un proceso.
- ▣ Un **input stream (flujo de entrada)** está ligado a alguna fuente de entrada para el proceso, Ej. teclado o socket.
- ▣ Un **output stream (flujo de salida)** está ligado a una salida del proceso, Ej. pantalla o socket.

Programación de sockets con TCP

Ejemplo aplicación cliente-servidor: A Mayúscula

- 1) Cliente lee líneas desde la entrada estándar (flujo **inFromUser**), las envía al servidor vía un socket (flujo **outToServer**)
- 2) El servidor lee líneas desde el socket
- 3) El servidor las convierte a mayúsculas, y las envía de vuelta al clientes
- 4) Cliente lee y muestra la línea modificada desde el socket (flujo **inFromServer**)





Interacción Cliente/servidor vía socket TCP

Servidor (corriendo en "hostname")

```
/* create socket,  
port=6789, for incoming request: */
```

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

```
/* wait for incoming connection request */
```

```
ConnectionSocket = welcomeSocket.accept()
```

```
/* read request from connectionSocket */
```

```
/* write reply to connectionSocket */
```

```
/*close connectionSocket*/
```

Cliente

```
/* create socket,  
connect to hostname, port=6789 */
```

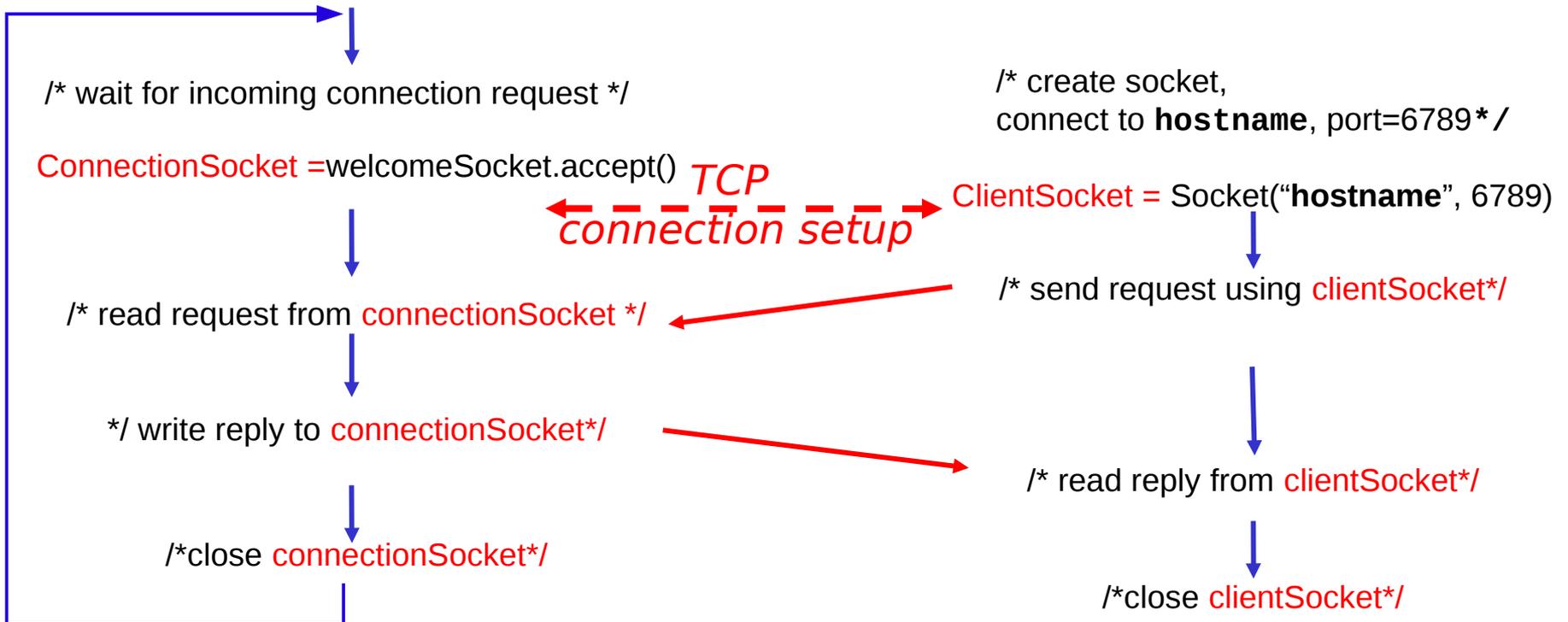
```
ClientSocket = Socket("hostname", 6789)
```

```
/* send request using clientSocket */
```

```
/* read reply from clientSocket */
```

```
/*close clientSocket*/
```

*TCP
connection setup*



Ejemplo: Cliente Java (TCP)

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

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

```
        String sentence;
        String modifiedSentence;
```

Traduce
hostname a IP
usando DNS

Crea
Flujo entrante

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

Crea
cliente socket,
conecta al servidor

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

Crea
Flujo de salida
Unido al socket

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

Ejemplo: Cliente Java (TCP), cont.

Flujo de entrada Unido al socket } Crea
BufferedReader inFromServer =
new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

Envía línea Al servidor } sentence = inFromUser.readLine();
outToServer.writeBytes(sentence + '\n');

Lee línea Desde el servidor } modifiedSentence = inFromServer.readLine();
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();

}
}

Ejemplo: Servidor Java (TCP)

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

```
class TCPServer {
```

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

```
        String clientSentence;  
        String capitalizedSentence;
```

Crea
Socket de
bienvenida
En puerto 6789

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

Crea socket de
conexión para
Contacto de clientes

```
        while(true) {
```

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

Crea flujo
De entrada unido
A socket

```
            BufferedReader inFromClient =
```

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

Ejemplo: Servidor Java (TCP), cont

Crea flujo de Salida unido al socket

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

Lee línea Desde el socket

```
clientSentence = inFromClient.readLine();
```

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

Escribe línea En socket

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

```
connectionSocket.close();
```

```
}  
}  
}
```

Fin del cuerpo del while,
Vuelve y espera
por la conexión de otro cliente
(un connectionSocket por línea de texto)

Capítulo 2: Capa Aplicación

- 2.1 Principios de la aplicaciones de red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo Electrónico
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P Compartición de archivos (Lo saltaremos)
- 2.7 Programación de sockets con TCP
- 2.8 Programación de sockets con UDP
- 2.9 Construcción de un servidor WEB

Programación de Socket *con UDP* User Datagram Protocol

UDP: no hay “conexión” entre cliente y servidor

- No hay handshaking (establecimiento de conexión)
- Tx explícitamente adjunta dirección IP y puerto de destino en cada paquete.
- Para responder se debe extraer dirección IP y puerto del Tx desde el paquete recibido

UDP: datos transmitidos pueden llegar fuera de orden o perderse.

Punto de vista de la aplicación

UDP provee transferencia no confiable de grupos de bytes (“datagramas”) entre cliente y servidor

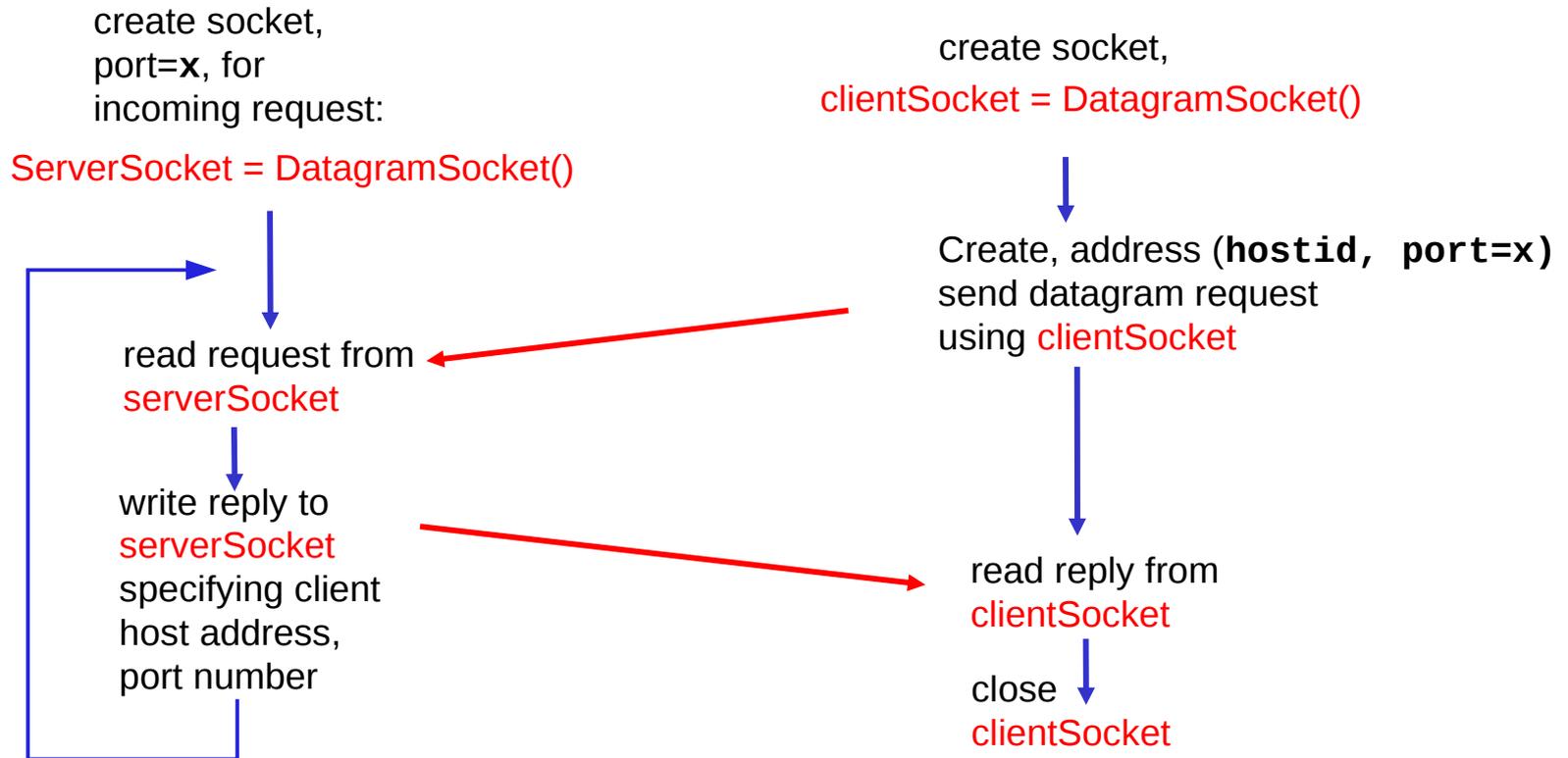
Distintos caminos al destino

Congestión

Interacción Cliente/servidor: UDP

Servidor (corriendo en `hostid`)

Cliente



Ejemplo: Cliente/servidor Java (UDP)

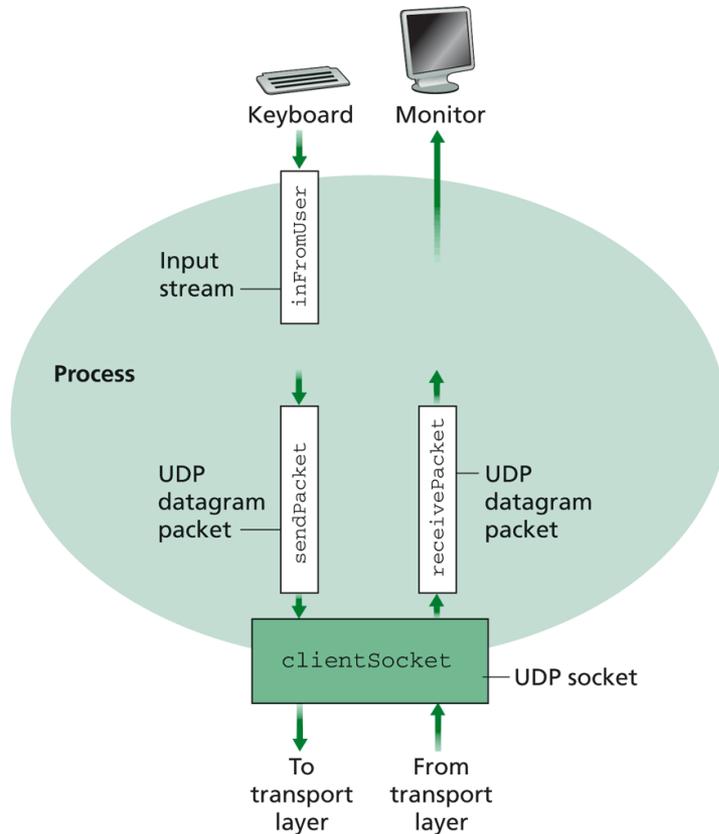


Figure 2.31 ♦ UDPClient has one stream; the socket accepts packets from the process and delivers packets to the process.

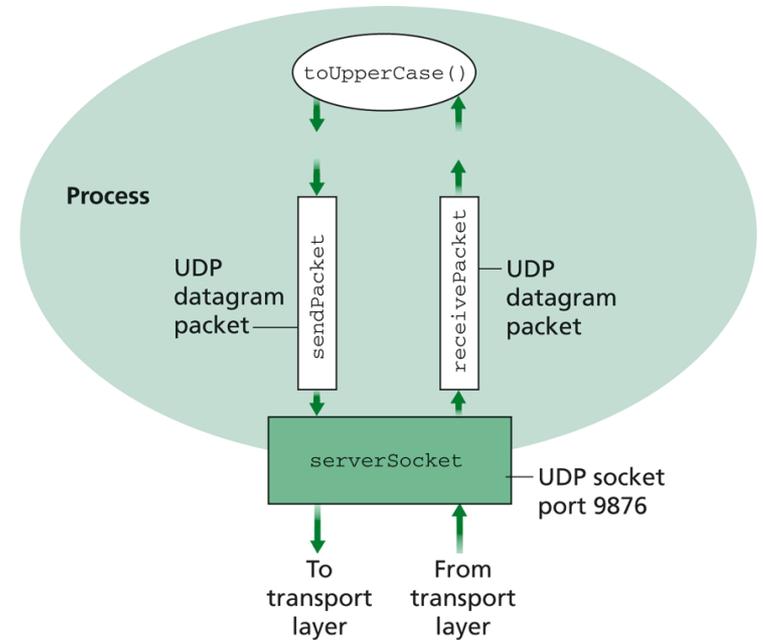


Figure 2.32 ♦ UDPServer has no streams; the socket accepts packets from the process and delivers packets to the process.

Ejemplo: Cliente Java (UDP)

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

```
class UDPClient {
    public static void main(String args[]) throws Exception
    {
```

Crea
input stream



```
BufferedReader inFromUser =
```

```
new BufferedReader(new InputStreamReader(System.in));
```

Crea
Socket cliente



```
DatagramSocket clientSocket = new DatagramSocket();
```

Traduce
hostname a IP
usando DNS



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

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

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

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

```
sendData = sentence.getBytes();
```

Ejemplo: Cliente Java (UDP), cont.

Crea datagrama con datos a enviar, largo, dir IP, puerto

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

Envía datagrama a servidor

```
clientSocket.send(sendPacket);
```

Lee datagrama desde servidor

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
clientSocket.receive(receivePacket);
```

```
String modifiedSentence = new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}  
}
```

Ejemplo: Servidor Java (UDP)

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

```
class UDPServer {
    public static void main(String args[]) throws Exception
    {
```

Crea
Socket de datagrama
en puerto 9876

```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

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

```
        while(true)
        {
```

Crea espacio para
recibir datagrama

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

Recibe
datagrama

```
            serverSocket.receive(receivePacket);
```

Ejemplo: Servidor Java (UDP), cont

```
String sentence = new String(receivePacket.getData());
```

Obtiene dir IP
puerto #, del
cliente

```
InetAddress IPAddress = receivePacket.getAddress();
```

```
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();
```

```
sendData = capitalizedSentence.getBytes();
```

Crea datagrama
a enviar a cliente

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

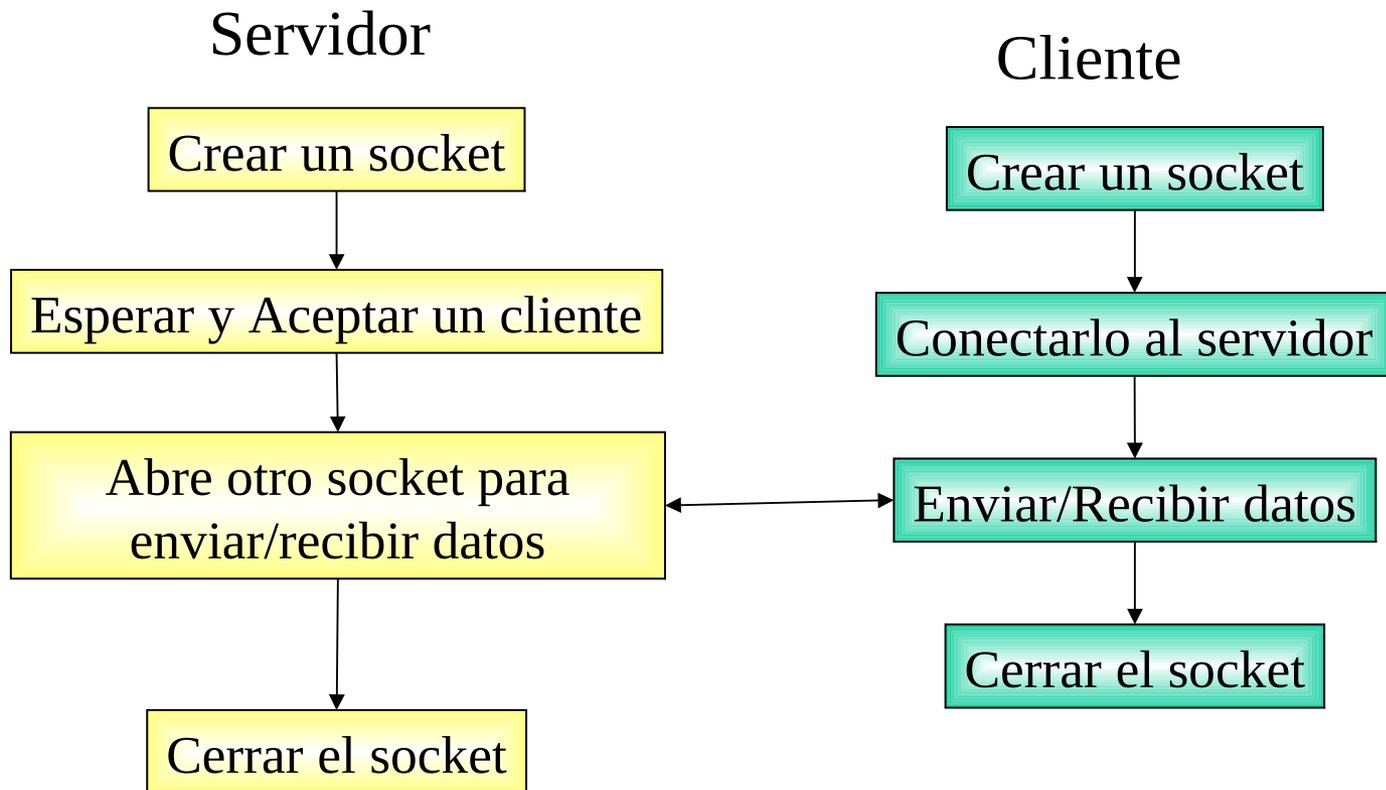
Envía el
datagrama a
través del socket

```
serverSocket.send(sendPacket);
```

```
}  
}  
}
```

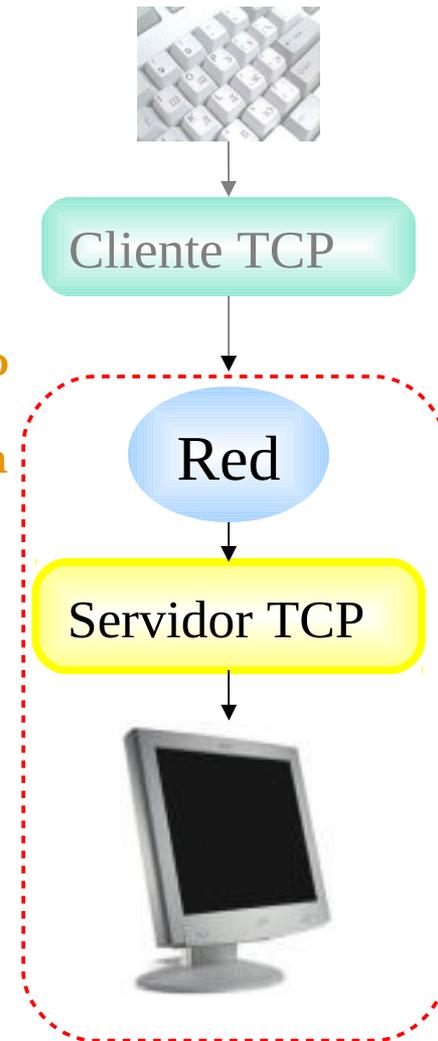
Término el cuerpo del while,
Vuelve a su inicio y espera
otro datagrama

Ejemplo 2 Cliente/Servidor TCP: Secuencia de Pasos en Java



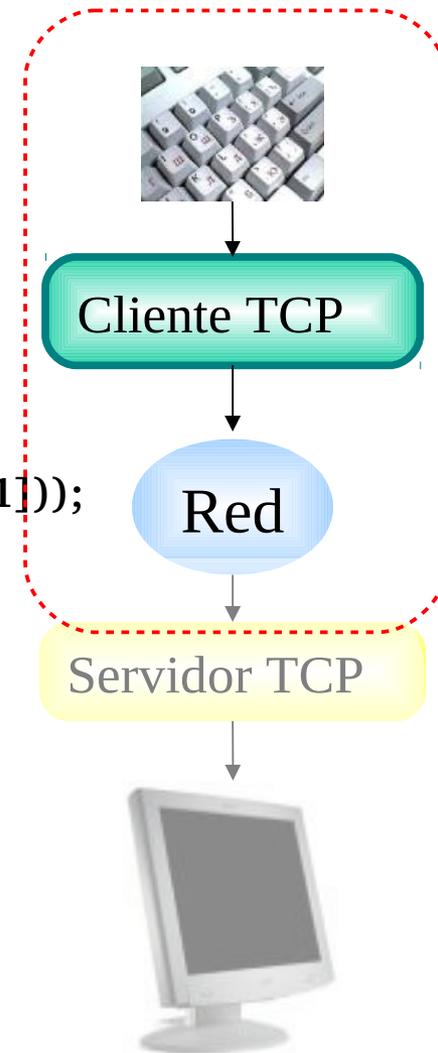
Servidor TCP en Java, Simple

```
import java.io.*;
import java.net.*;
class TCPserver {
    public static void main (String argv[]) throws Exceptio {
        String line; // Almacena lo recibido
        //welcomeSocket es el socker servidor que acepta la conexión
        ServerSocket welcomeSocket = new ServerSocket(
            Integer.parseInt(argv[0]));
        // connectionSocket es aquel que atiende a un cliente específico
        Socket connectionSocket = welcomeSocket.accept();
        // Esta concatenación de objetos adaptadores permite la lectura
        // simple de datos desde el socket para ese cliente.
        BufferedReader inFromClient = new BufferedReader(
            new InputStreamReader(connectionSocket.getInputStream()));
        // Recibe datos y los envia a pantalla.
        do {
            line=inFromClient.readLine();
            System.out.println(line);
        } while(!line.equals("quit"));
        // Cerramos ambos sockets
        connectionSocket.close();
        welcomeSocket.close();
    }
}
```



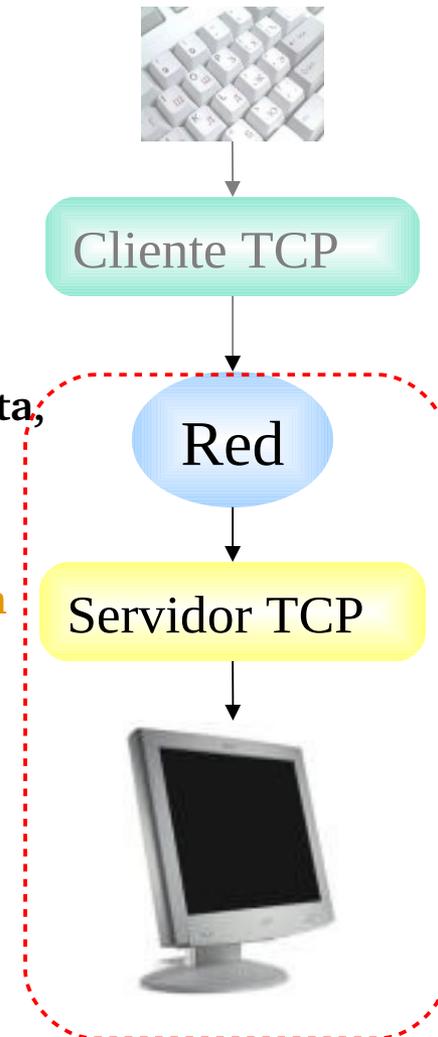
Cliente TCP en Java , Simple

```
import java.io.*;
import java.net.*;
class TCPclient {
    public static void main (String argv[]) throws Exception {
        String line; // Almacena lo digitado
        // Concatenación de objetos adaptadores para la lectura
        // simple de teclado.
        BufferedReader inFromUser = new BufferedReader(
            new InputStreamReader(System.in));
        // Socket en el cliente para enviar datos al servidor.
        Socket clientSocket = new Socket(argv[0],Integer.parseInt(argv[1]));
        // Concatenación de objetos adaptadores para la escritura
        // o envío de datos en forma simple a través del socket.
        DataOutputStream outToServer = new DataOutputStream(
            clientSocket.getOutputStream());
        // Lectura de teclado y envío de datos al servidor.
        do {
            line=inFromUser.readLine();
            outToServer.writeBytes(line+ '\n');
        } while(!line.equals("quit"));
        // Cerramos el socket y con ello también la conexión.
        clientSocket.close();
    }
}
```



Servidor UDP en Java, Simple

```
import java.io.*;
import java.net.*;
class UDPserver {
    public static void main (String argv[] ) throws Exception {
        // construimos un socket ligado a un puerto. Pasa a ser servidor.
        DatagramSocket serverSocket = new DatagramSocket(
            Integer.parseInt(argv[0]));
        // buffer que contendrá los datos recibidos
        byte[] receiveData = new byte[256];
        // Datagrama que recibe lo enviado por el cliente.
        DatagramPacket receivePacket = new DatagramPacket (receiveData,
            receiveData.length);
        String line; // almacenará la línea enviada.
        do {
            serverSocket.receive(receivePacket); // Recibimos un datagrama
            // y extraemos de él la línea enviada desde la posición 0
            // al largo de datos recibidos.
            line = new String(receivePacket.getData(), 0,
                receivePacket.getLength());
            System.out.print(line); // muestra línea en pantalla.
        } while (!line.equals("quit" + "\n"));
        // Cerramos ambos sockets
        serverSocket.close();
    }
}
```

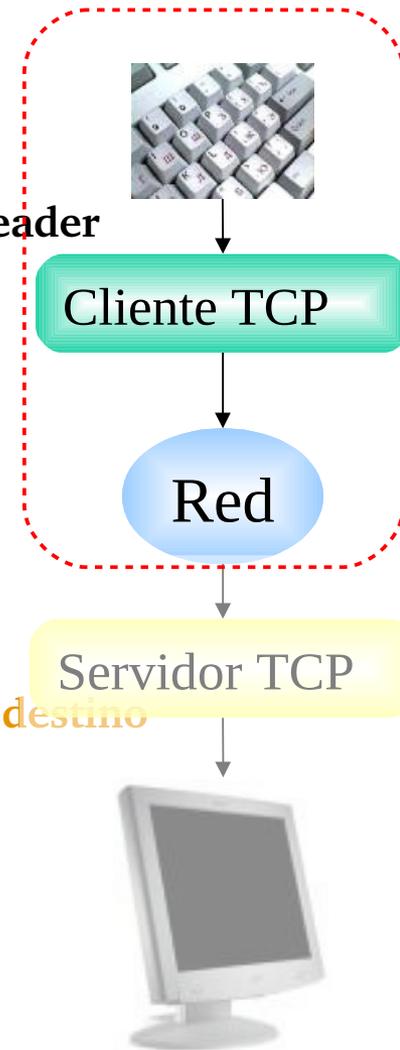


Cliente UDP en Java, Simple

```
import java.io.*;
import java.net.*;
class UDPclient {
    public static void main (String argv[]) throws Exception {
        // Concatenación de objetos adaptadores para la lectura
        // simple de teclado.
        BufferedReader inFromUser=new BufferedReader(new InputStreamReader
            (System.in));

        // Socket en el cliente para enviar datos al servidor.
        DatagramSocket clientSocket = new DatagramSocket();
        // Creamos objeto con dirección IP destino
        InetAddress IPAddress = InetAddress.getByAddress(argv[0]);
        // Puerto a definir en el datagrama a enviar
        int port = Integer.parseInt(argv[1]);
        String line; // línea a leer de teclado
        do {
            line = inFromUser.readLine()+"\n";
            byte[] sendData = line.getBytes(); // sacamos los bytes del string
            // se construye el Datagrama UDP con los datos, dirección y puerto destino
            DatagramPacket sendPacket = new DatagramPacket(sendData,
                sendData.length,IPAddress,port);

            // enviamos el datagrama
            clientSocket.send(sendPacket);
        }while (!line.equals("quit"+'\n'));
        // Cerramos el socket
        clientSocket.close();
    }
}
```



Capítulo 2: Capa Aplicación

- 2.1 Principios de la aplicaciones de red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo Electrónico
 - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P Compartición de archivos (Lo saltaremos)
- 2.7 Programación de sockets con TCP
- 2.8 Programación de sockets con UDP
- 2.9 Construcción de un servidor WEB

Construyendo un servidor Web simple

- Maneja una petición HTTP
- Acepta la petición
- Analiza cabecera
- Obtiene archivo pedido de su sistema de archivos
- Crea mensaje HTTP de respuesta:
 - líneas cabecera + archivo
- Manda respuesta al cliente
- Después de crear este servidor, tú le puedes pedir un archivo usando un browser (eg Mozilla, Netscape o IE explorer)
- Este es un ejercicio principalmente de programación.
- Ver texto guía para más detalles

Resumen de Capa aplicación

Hemos cubierto varias aplicaciones de red

- Arquitectura de la aplicaciones
 - cliente-servidor
 - P2P
 - híbridos
- Servicios requeridos por aplicaciones:
 - confiabilidad, ancho de banda, retardo
- Modelo de servicio de transporte en Internet
 - Confiable y orientada a la conexión: TCP
 - No confiable, datagramas: UDP
- Protocolos específicos:
 - HTTP
 - FTP
 - SMTP, POP, IMAP
 - DNS
- Programación de sockets
- Un servidor web simple (ver texto)

Resumen de Capa aplicación

Lo más importante aprendido sobre *protocolos*

- ▣ Intercambio de mensajes típicos requerimiento/respuesta:
 - ▣ cliente requiere info o servicio
 - ▣ servidor responde con datos, código de estatus
- ▣ Formato de mensajes:
 - ▣ encabezado: campos dando info sobre datos
 - ▣ datos: info siendo comunicada
- ▣ Mensajes de control vs. datos
 - in-band, out-of-band
- ▣ Centralizado vs. descentralizado
- ▣ Sin estado vs. con estado
- ▣ Transferencia confiable vs. Transferencia no confiable
- ▣ “la complejidad es puesta en los bordes de la red (las aplicaciones)”
Distinto a sistema telefónico clásico.