Capítulo 2: Capa Aplicación - IV

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.

- Material del curso anterior ELO322 del Prof. Tomás Arredondo V.

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

P2P file sharing (compartición de Archivos)

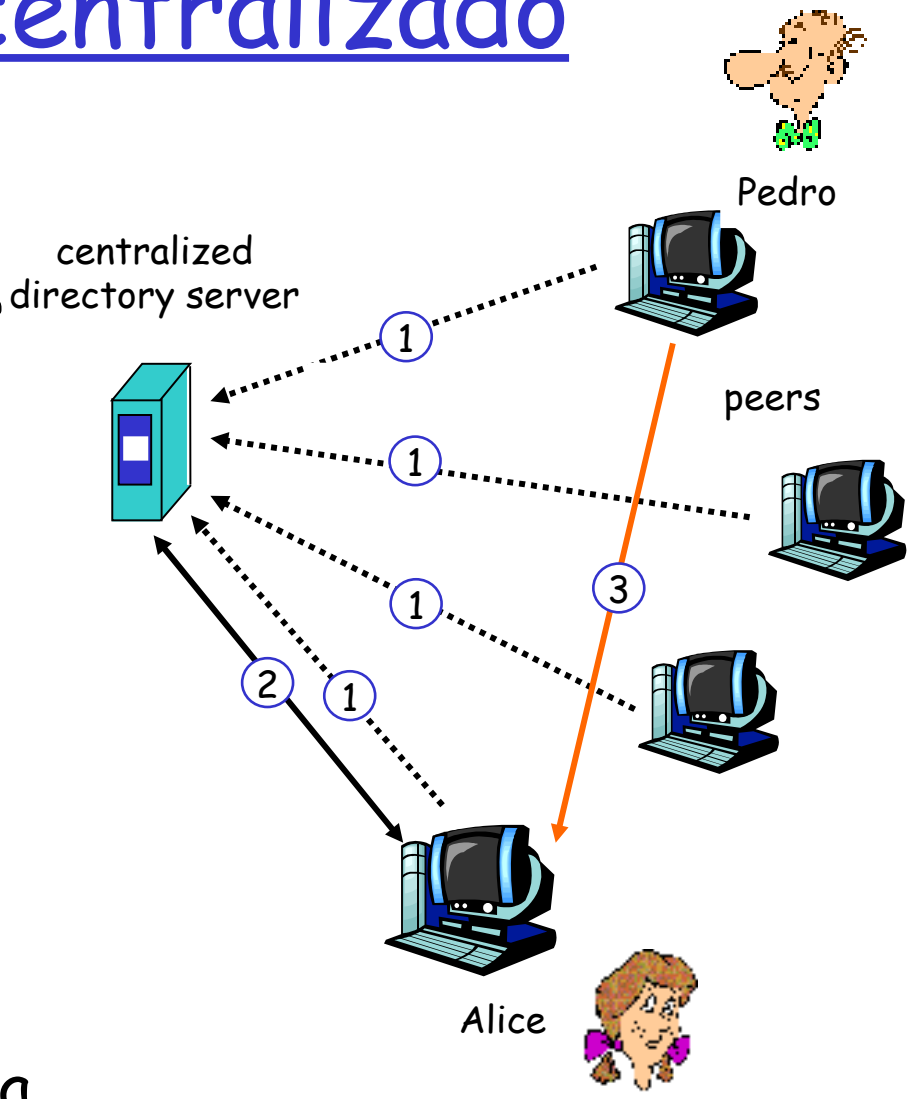
Ejemplo

- Alice ejecuta una aplicación cliente en su notebook
- Intermitentemente se conecta al Internet; recibe una nueva dirección IP en cada conexión
- Pide canción "Hey Jude"
- Aplicación muestra otros pares que tienen una copia de "Hey Jude".
- Alice elige a uno de los pares, Pedro
- Archivo es copiado del PC de Pedro al PC de Alice notebook: HTTP
- Mientras que Alice lo baja, otros usuarios bajan música desde el PC de Alice.
- El PC de Alice es un cliente Web y también temporalmente un servidor Web.
- Todos los pares puedes ser servidores => altamente escalable!

P2P: directorio centralizado

Diseño original de "Napster"

1) Cuando un terminal napster se conecta, él informa a un servidor central:
   - dirección IP
   - música que tiene
2) Alice pregunta por "Hey Jude", se entera lo tiene Pedro
3) Alice pide luego el archivo a Pedro directamente



P2P: problemas con directorio centralizado

- Punto individual de falla
- Cuello de botella a la capacidad (performance)
- Problemas legales con música (Copyright infringement)

la transferencia de archivos es descentralizada pero la localización de contenido (archivos) es altamente centralizado

Inundación de preguntas (Query flooding): Gnutella

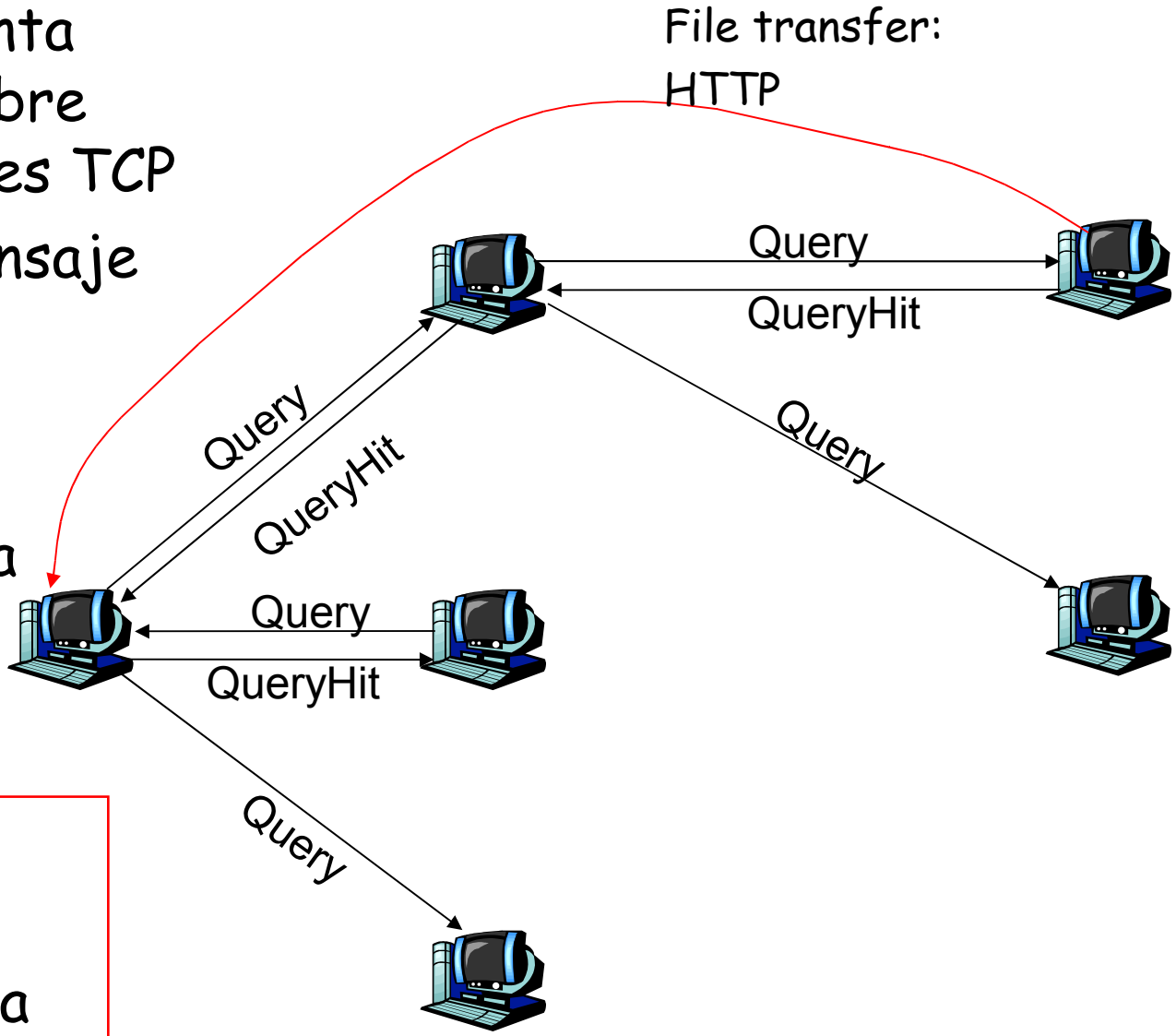
- Completamente distribuido
  - sin servidor central
- Protocolo de dominio público
- Muchos clientes Gnutella implementan el protocolo

Red sobrepuesta: grafo

- Hay enlace entre pares X e Y si hay una conexión TCP
- Todos los pares activos y sus enlaces forman la red sobrepuesta (overlay net)
- Cada enlace no es un enlace físico sino conceptual
- Un par típicamente va a estar conectado a < 10 vecinos en su red sobrepuesta

Gnutella: protocolo

- Mensaje de pregunta (Query) mandado sobre conexiones existentes TCP
- Pares reenvían mensaje de pregunta (Query message)
- Resultado positivo (QueryHit) se manda por ruta reversa



Escalable:

inundación de mensajes limitada

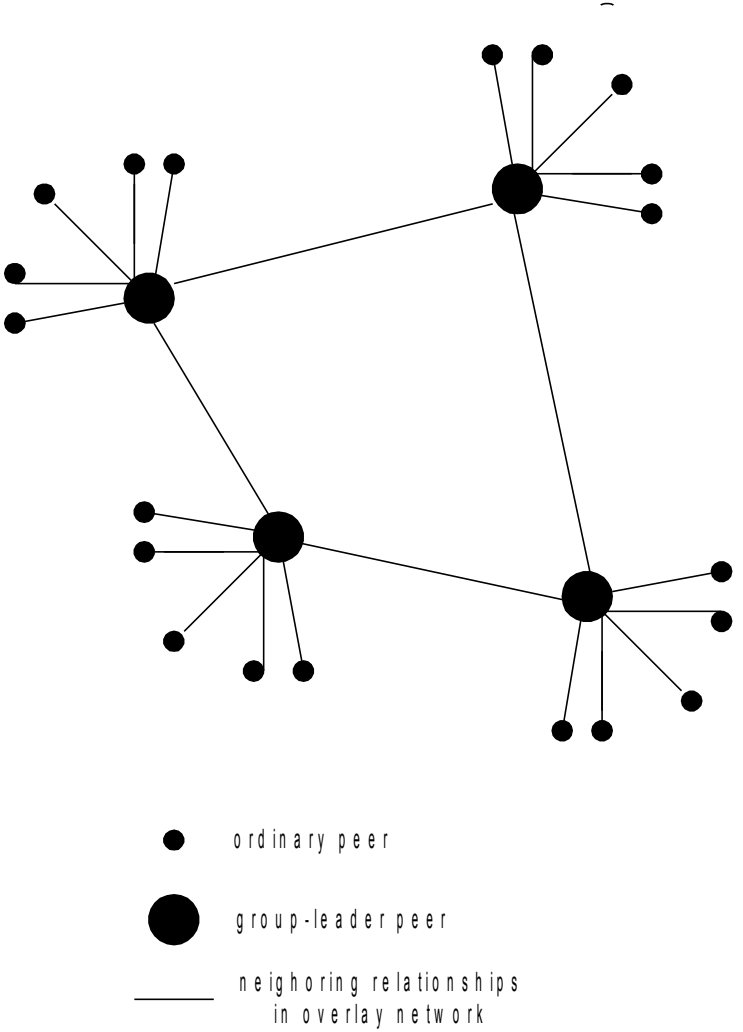
Gnutella: Conectarse a Pares

1. Nodo X debe encontrar otro par en la red Gnutella: usa lista de pares candidatos
2. X trata secuencialmente de conectarse vía TCP con pares en su lista hasta hacer una conexión con Y
3. X manda mensaje Ping a Y; Y reenvía mensaje Ping
4. Todos los pares que reciben el mensaje Ping responden con mensaje Pong

- X recibe muchos mensajes Pong. Ahora él puede establecer conexiones TCP adicionales.

Explotando heterogeneidad: KaZaA

- Protocolo no público
- Cada nodo es un líder de grupo o asignado a un líder de grupo
  - Conexión TCP entre nodo y líder de grupo
  - Conexiones TCP entre pares de lideres de grupo
- Líder de grupo sabe los contenidos (archivos) de todos sus hijos



KaZaA: Búsquedas

- Cada archivo tiene un hash y un descriptor (incluye el nombre del archivo y descripción en texto del objeto)
- Cliente manda una pregunta usando palabras claves a su líder de grupo (él busca en el descriptor)
- Líder de grupo responde con aciertos:
  - Para cada acierto: metadatos, hash, direccion IP
- Si un líder de grupo reenvía la búsqueda a otros lideres de grupo, esos lideres contestan con aciertos (usando ruta inversa red sobrepuesta)
- Cliente selecciona archivos para bajar
  - Mensajes HTTP usando hash como identificador son mandados a pares que contienen archivo deseado

Trucos KaZaA

- Limitación para subidas (uploads) (y downloads?) simultaneas
- Encolamiento de peticiones
- Prioridades basadas en incentivos a mejores usuarios (los que suben más archivos a la red)
- Bajada de datos para un archivo en paralelo (puede usar múltiples conexiones HTTP a diferentes pares para el mismo archivo)

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 liberado 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

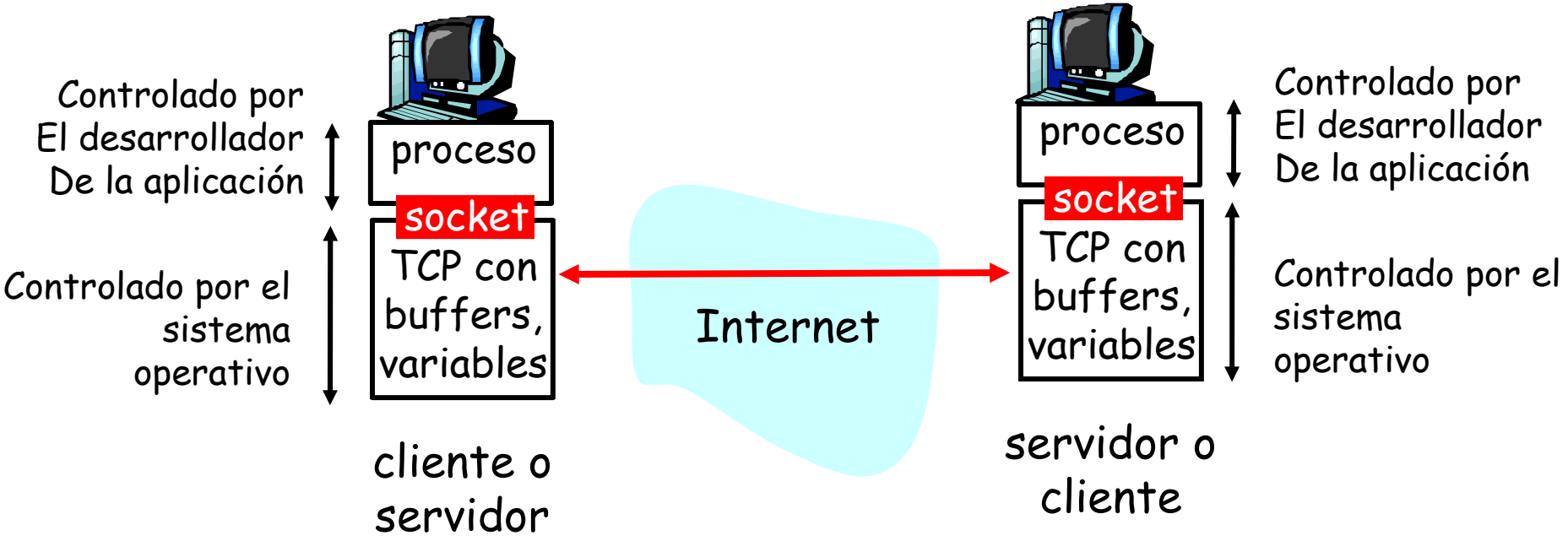
Son *locales al host*, *creados por la aplicación*, *Es una interfaz controlada por el OS* (una "puerta") a través de la cual el proceso aplicación puede tanto enviar como recibir mensajes a/desde el otro proceso de la aplicación

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 comunique 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

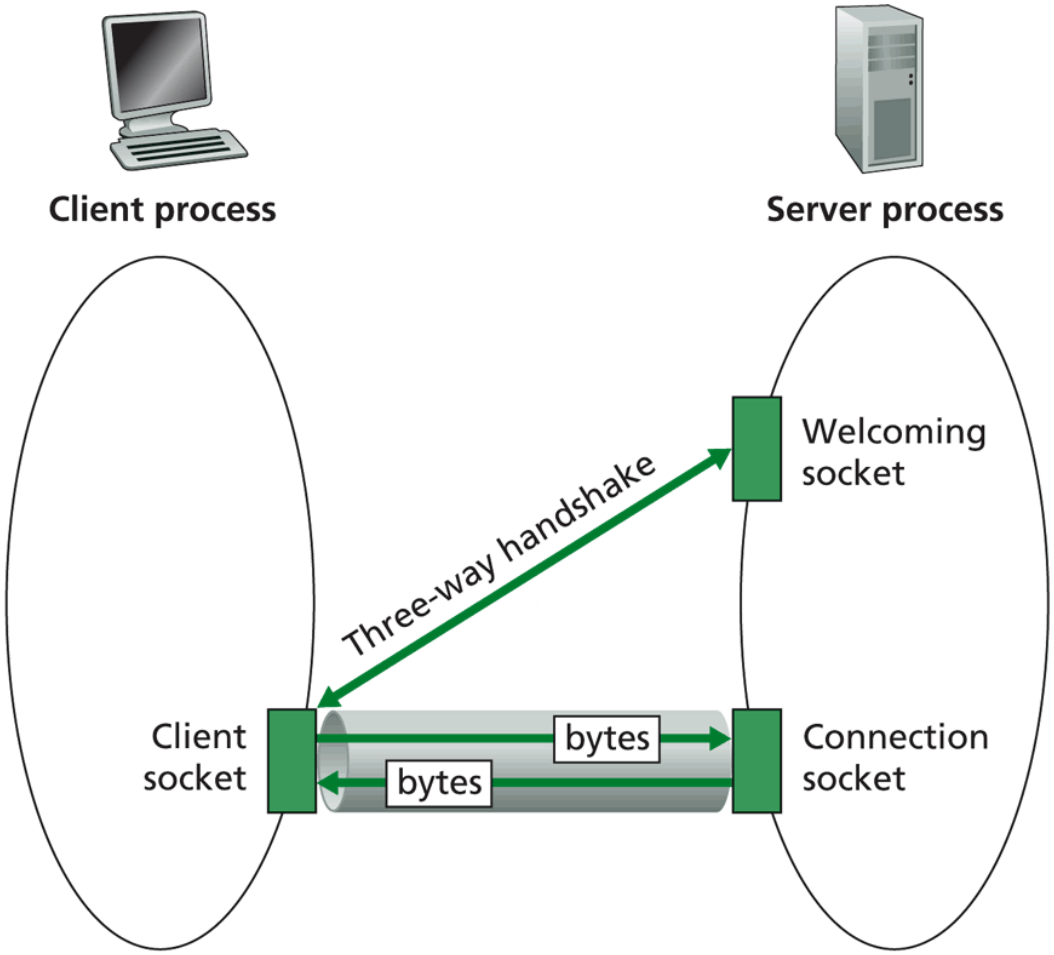


Figure 2.27 ♦ Client socket, welcoming socket, and connection socket

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, eg, teclado o socket.
- Un output stream (flujo de salida) está ligado a una salida del proceso, eg, pantalla o socket.

Programación de sockets con TCP

Ejemplo aplicación cliente-servidor:

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`**)

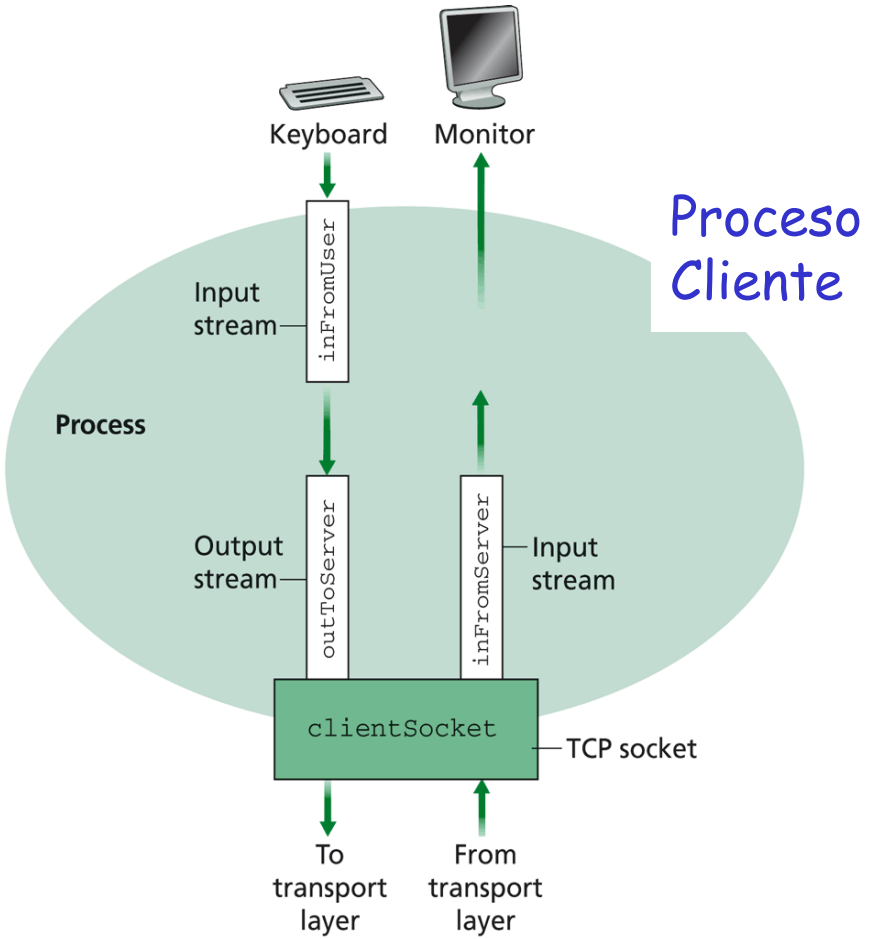
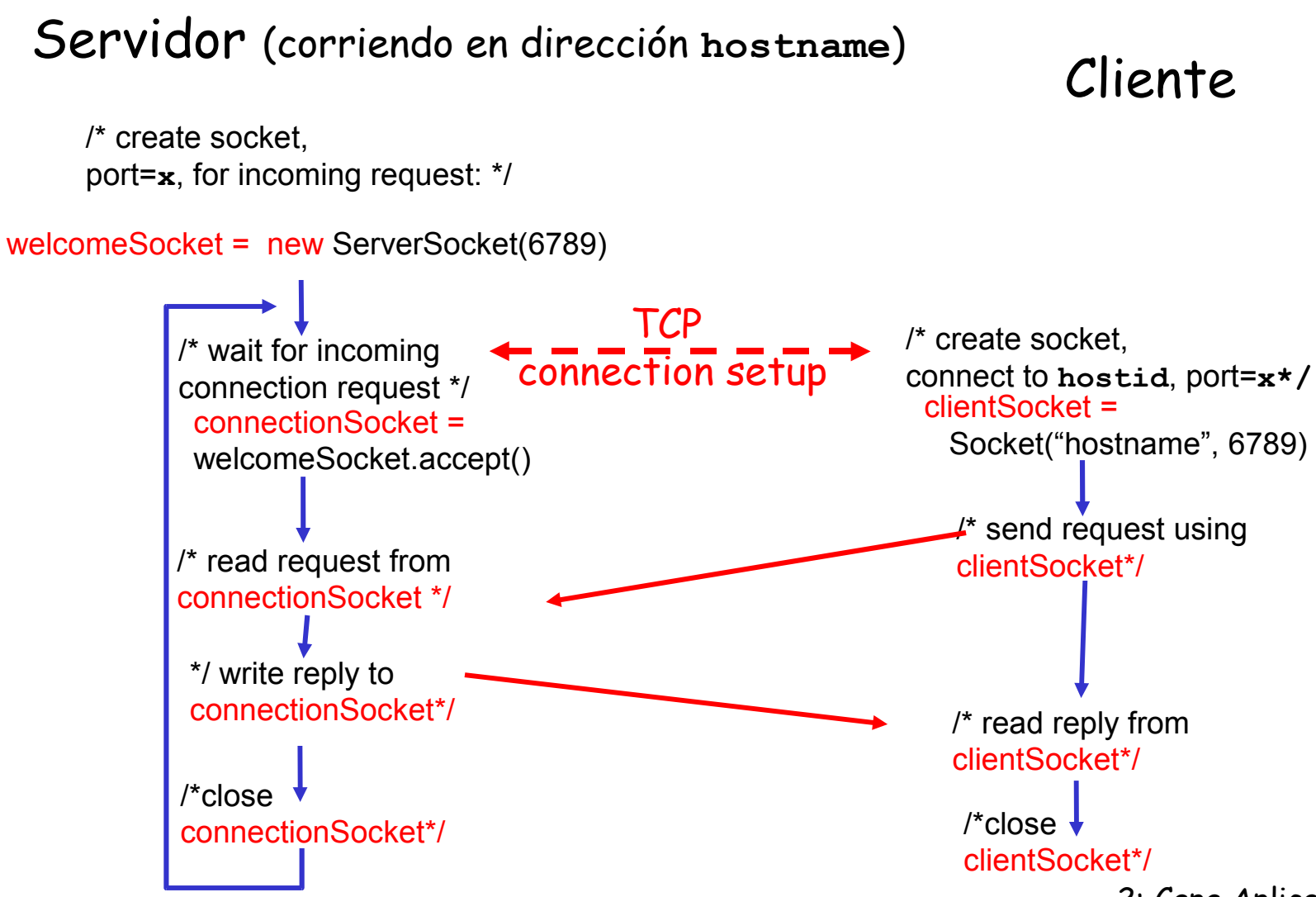


Figure 2.29 ♦ `TCPClient` has three streams through which characters flow.

Interacción Cliente/servidor vía socket TCP



Ejemplo: Cliente Java (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());
```

Traduce hostname a IP usando DNS

Crea Flujo entrante

Crea cliente socket, conecta al servidor

Crea Flujo de salida Unido al socket

Ejemplo: Cliente Java (TCP), cont.

Crea
Flujo de entrada
Unido al socket

```
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();

    }
}
```

Envía línea
Al servidor

Lee línea
Desde el servidor

Ejemplo: Servidor Java (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()));
```

Crea Socket de bienvenida En puerto 6789

Crea socket de conexión para Contacto de clientes

Crea flujo De entrada unido A socket

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);
        }
    }
}
```

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

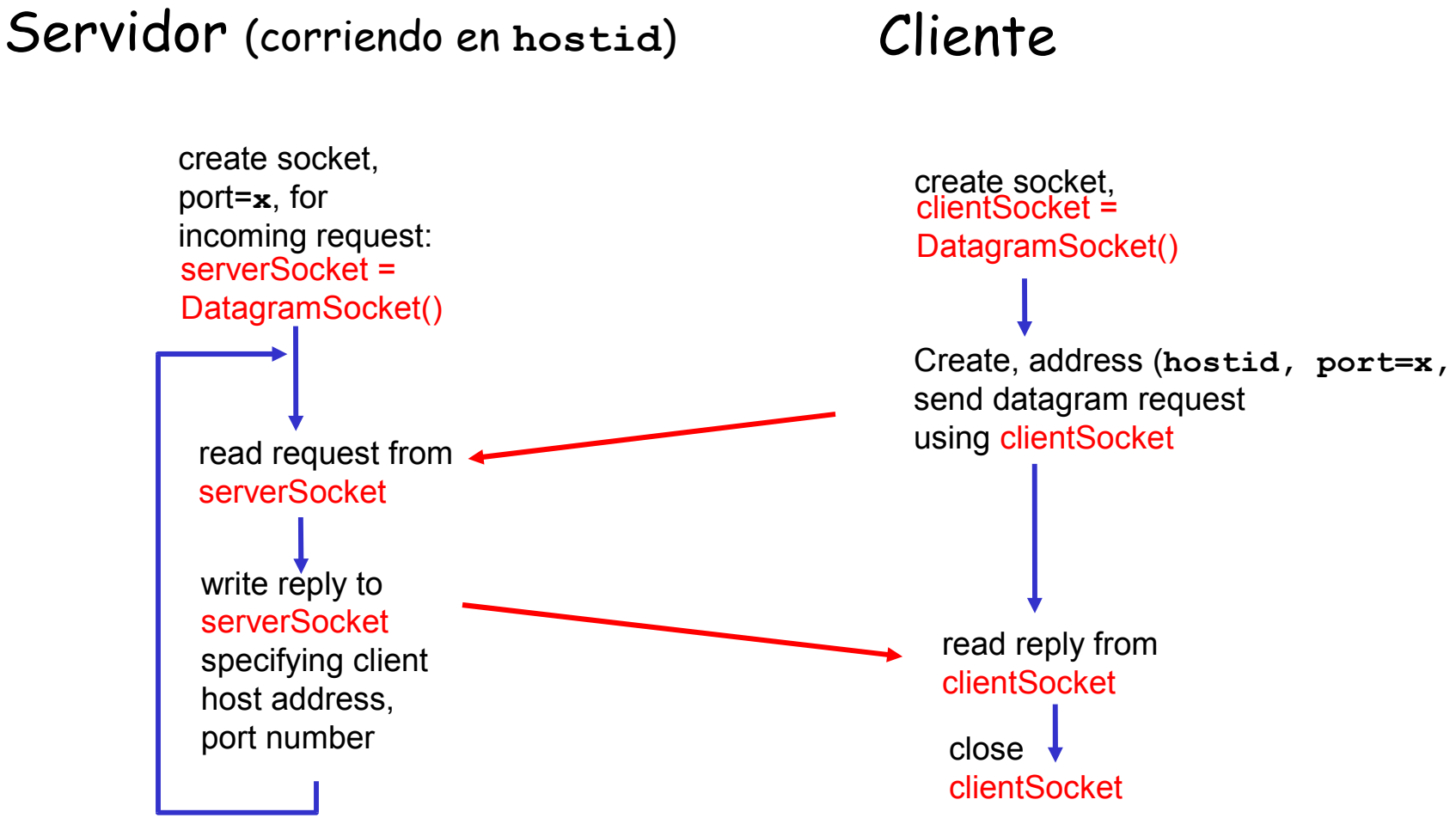
- No hay handshaking (establecimiento de conexión)
- Tx explícitamente adjunta dirección IP y puerto de destino en cada paquete.
- Para responder se debe extraer dirección IP y puerto del Tx desde el paquete recibido

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

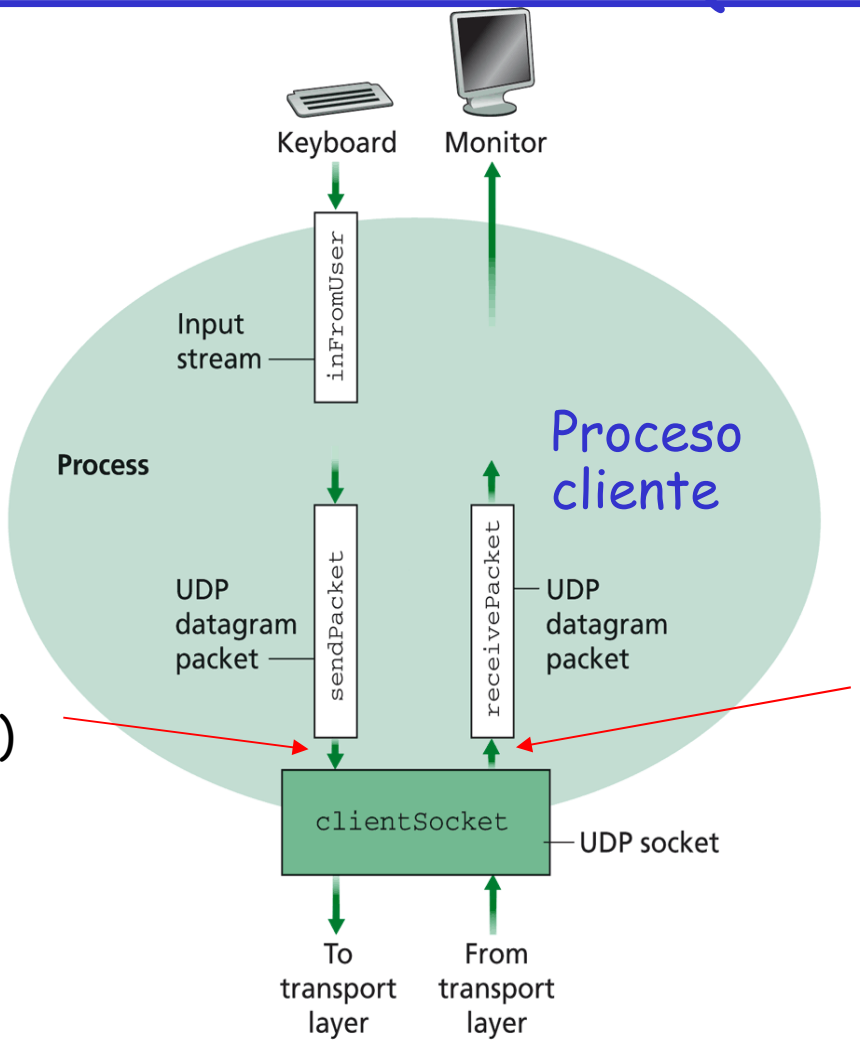
Punto de vista de la aplicación

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

Interacción Cliente/servidor: UDP



Ejemplo: Cliente Java (UDP)



Output: envía paquetes (envía "flujo de bytes")

Input: recibe paquetes (recibe "flujo de bytes")

Figure 2.31 ♦ `UDPClient` has one stream; 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
   {
      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();
```

Crea input stream

Crea Socket cliente

Traduce hostname a IP usando DNS

Ejemplo: Cliente Java (UDP), cont.

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

Envía datagrama a servidor

Lee datagrama desde servidor

```
                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();
                }

            }
```

Ejemplo: Servidor Java (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);
```

Crea Socket de datagrama en puerto 9876

Crea espacio para recibir datagrama

Recibe datagrama

Ejemplo: Servidor Java (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);
        }
    }
}
```

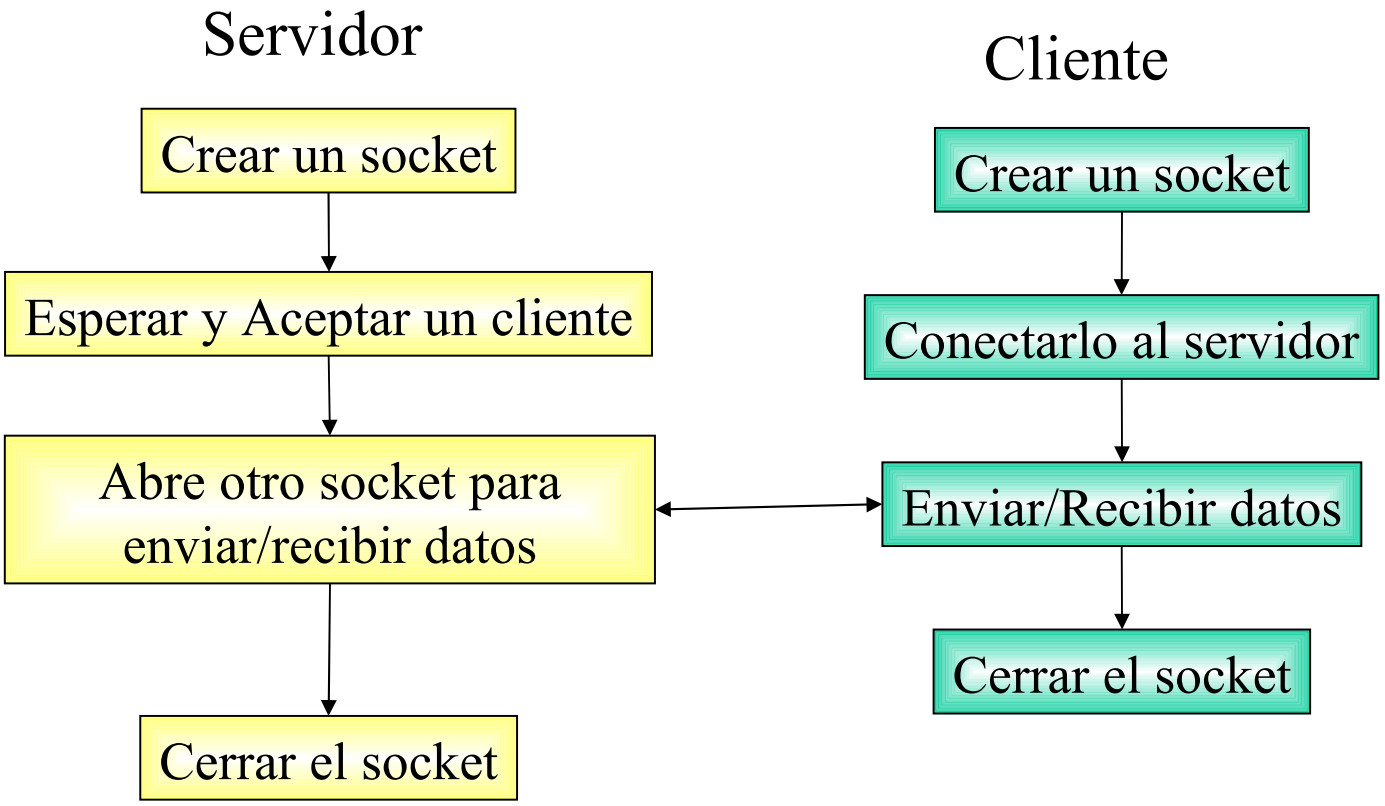
Obtiene dir IP puerto #, del cliente

Crea datagrama a enviar a cliente

Envía el datagrama a través del socket

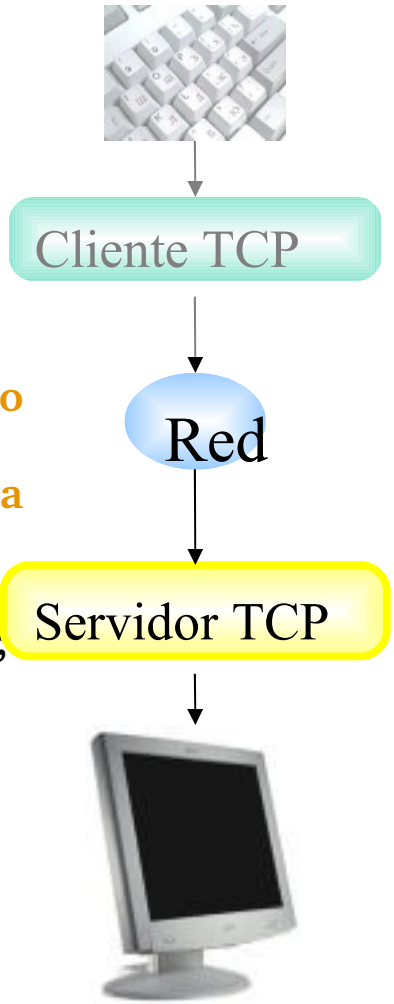
Término el cuerpo del while, Vuelve a su inicio y espera otro datagrama

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



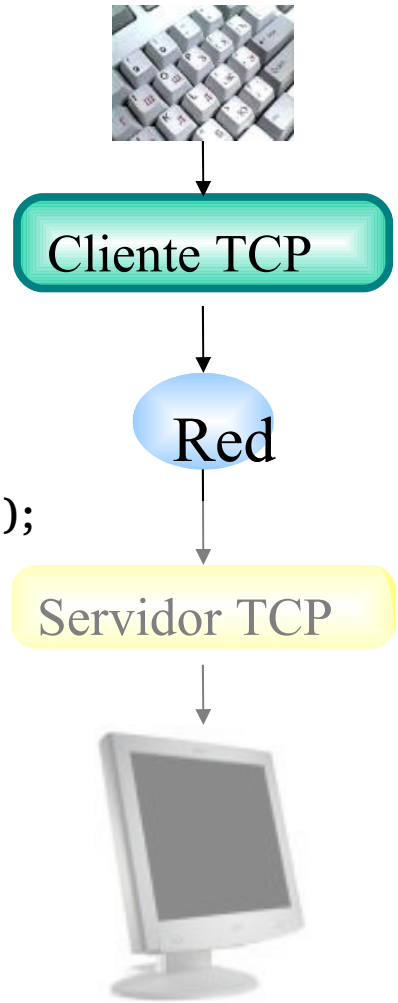
Servidor TCP en Java, Simple

```
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();
  }
}
```



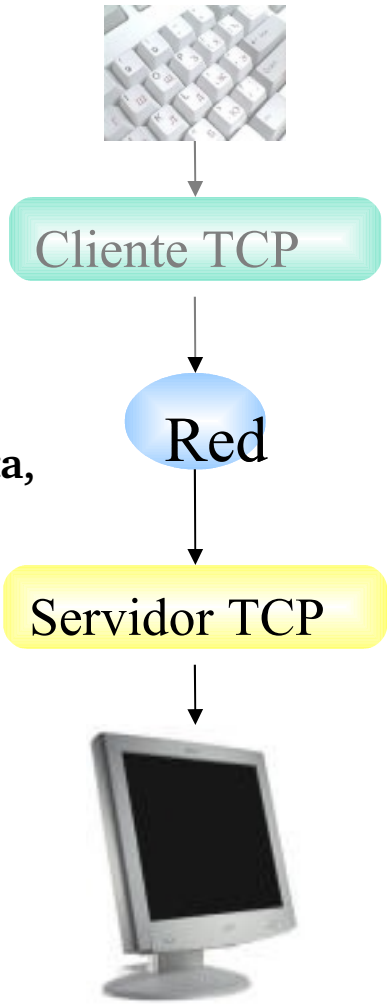
Cliente TCP en Java , Simple

```
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 envio 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();
  }
}
```



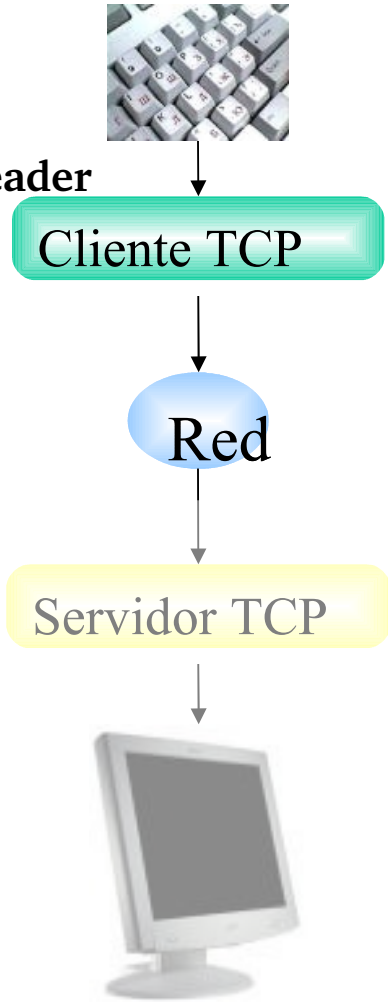
Servidor UDP en Java, Simple

```
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 linea 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.getByName(argv[0]);
     // Puerto a definir en el datagrama a enviar
    int port = Integer.parseInt(argv[1]);
    String line;  // linea 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 (parses header)
- Obtiene archivo pedido de su sistema de archivos (file system)
- Crea mensaje HTTP de respuesta:
  - líneas cabecera + archivo
- Manda respuesta al cliente

- Después de crear el servidor, tu puedes pedir un archivo usando un browser (eg Mozilla, Netscape o IE explorer)
- Ver texto 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.