**Instituto Politécnico Nacional**

Escuela Superior de Cómputo

Reporte Práctica 1

**Integrantes del Equipo:**

Barrera Pérez Carlos Tonatihu

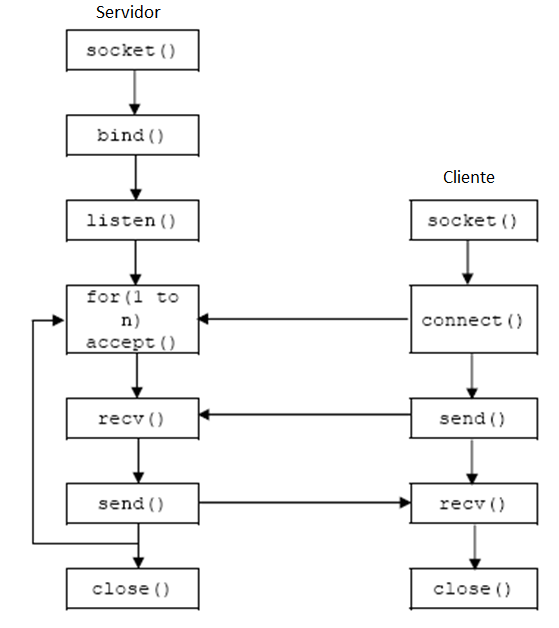
Castillo Reyes Juan Daniel

**Grupo:** 2CM11

**Unidad de Aprendizaje:** Aplicaciones para Comunicaciones en Red

**Profesor:** Axel Ernesto Moreno Cervantes

Introducción



**Sockets TCP**

Una conexión TCP es un canal abstracto bidireccional cuyos extremos están identificados por una dirección IP y un número de puerto. Antes de iniciar una comunicación una conexión TCP debe comenzar por el cliente TCP enviando una solicitud de conexión al servidor. En el caso de Java una instancia de la clase *ServerSocket* es la encargada de escuchar estas solicitudes de conexión y crear una nueva instancia de la clase *Socket* para manejar esta conexión.

**Cliente**

Los pasos que realiza el cliente para iniciar una conexión en el caso de Java son los siguientes:

1. Crear una instancia de la clase Socket con esto se establece una conexión con un host y un puerto.
2. Comunicarse usando flujos de entrada/salida (*InputStream/OutputStream*)
3. Finalmente, cerrar la conexión con el servidor *close()*.

**Servidor**

El objetivo del servidor es configurar la comunicación y esperar a los clientes. La forma en la que lo hace es la siguiente:

1. Crear una instancia de *ServerSocket* mediante la especificación de un puerto. Al hacer esto esperaremos por conexiones al puerto especificado.
2. Creamos una instancia de Socket al usar el método *accept()* de *ServerSocket*, nos comunicamos con el cliente mediante los flujos de entrada/salida. Terminamos la conexión con el cliente al llamar el método *close()* de *Socket*.
3. Repetimos el paso anterior las veces que sean necearías.

Desarrollo

Esta práctica consistió en el desarrollo de un programa que permitiera el envío de archivos utilizando sockets de flujo debido a que se trabajan con archivos es indispensable el uso de una interfaz gráfica con la cual se pudiera interactuar.

Además, la interfaz debería permitir el arrastrar archivos y enviarlos, sin olvidar que también se tiene que poder arrastrar carpetas y su contenido debería ser creado en el servidor de la misma forma en la que se encontraba en el cliente.

El código principal de esta práctica se encuentra en dos clases, una que se encarga del envío de la información (socket cliente) y otra que maneja el almacenamiento de los archivos (socket servidor). Una clase más fue utilizada para poder implementar la función Drag and Drop.

**Clase SocketEnvio.java**

|  |
| --- |
| **package** sockets;  **import** **java.io.\***;  **import** **java.net.Socket**;  */\*\**  *\* @author tona*  *\*/*  **public** **class** **SocketEnvio** {  **private** **final** String host;  **private** **final** int port;  **public** SocketEnvio(String host, int port) {  **this**.host = host;  **this**.port = port;  }  **public** void enviarArchivo(File file, String destino) **throws** IOException {  Socket cl = **new** Socket(host, port);  String nombre = file.getName();  long tam = file.length();  long enviados = 0;  int porcentaje;  int n;  String ruta = file.getAbsolutePath();  System.out.println("Conexion establecida");  DataOutputStream dos = **new** DataOutputStream(cl.getOutputStream());  DataInputStream dis = **new** DataInputStream(**new** FileInputStream(ruta));  *// Usamos destino para ir almacenando la ruta del archivo/carpeta*  dos.writeUTF(destino);  dos.flush();  *// Despues mandamos el nombre del archivo*  dos.writeUTF(nombre);  dos.flush();  *// Y finalmente su tamaño*  dos.writeLong(tam);  dos.flush();  System.out.format("Enviando el archivo: %s...\n", nombre);  System.out.format("Que esta en la ruta: %s\n", ruta);  **while** (enviados < tam) {  byte[] b = **new** byte[1500];  n = dis.read(b);  dos.write(b, 0, n);  dos.flush();  enviados = enviados + n;  porcentaje = (int) (enviados \* 100 / tam);  System.out.println("\rSe ha transmitido el: " + porcentaje + "% ...");  }  System.out.println("Archivo enviado");  cl.close();  dos.close();  dis.close();  }  *// el parametro destino nos permite guardar la ubicacion del archivo*  **public** void enviarCarpetas(File carpeta, String destino) **throws** IOException {  System.out.format("Carpeta %s con los archivos:\n", carpeta.getName());  **if** (destino.equals("")) destino = carpeta.getName(); *// evita que se cree en c:\\*  **else** destino = destino + "\\" + carpeta.getName(); *// concatenar la ruta de los archivos*  **for** (File file : carpeta.listFiles()) {  **if** (file.isDirectory()) enviarCarpetas(file, destino);  **else** enviarArchivo(file, destino);  }  }  } |

La tarea de esta clase consiste en enviar los archivos que son seleccionados en el resto de clases ya sea utilizando el método enviarArchivo() o el método enviarCarpeta(). También crea el socket que se utilizara para la transferencia.

**Clase Servidor.java**

|  |
| --- |
| **package** sockets;  **import** **java.io.\***;  **import** **java.net.ServerSocket**;  **import** **java.net.Socket**;  */\*\**  *\* @author tona*  *\*/*  **public** **class** **Servidor** {  **private** **static** **final** int PUERTO = 9999;  **public** **static** void main(String[] args) **throws** IOException {  ServerSocket s = **new** ServerSocket(PUERTO);  s.setReuseAddress(**true**);  System.out.println("Servicio iniciado...");  **for** (; ; ) {  System.out.println("Eperando conexion...");  Socket cl = s.accept();  System.out.format("Cliente conectado desde: %s:%s\n", cl.getInetAddress(),  cl.getPort());  DataInputStream dis = **new** DataInputStream(cl.getInputStream());  String ruta = dis.readUTF();  String nombre = dis.readUTF();  nombre = crearDirectorio(nombre, ruta);  escribirArchivo(nombre, dis);  System.out.println("¡Archivo recibido!\n");  dis.close();  cl.close();  }  }  **private** **static** String crearDirectorio(String nombre, String ruta) {  *// Verificamos si el archivo se crea en la raiz o en alguna carpeta*  **if** (!ruta.equals("")) {  File directorio = **new** File(ruta); *// Directorio de almacenamiento*  *// Si no existe lo creamos*  **if** (!directorio.exists()) {  **try** {  **if** (directorio.mkdir()) System.out.println("Carpeta creada");  **else** System.out.println("No se pudo crear la carpeta");  } **catch** (SecurityException se) {  se.printStackTrace();  }  }  *// La ubicacion final de nuestro archivo*  nombre = ruta + "\\" + nombre;  }  **return** nombre;  }  **private** **static** void escribirArchivo(String nombre, DataInputStream dis) {  System.out.format("Escribiendo el archivo: %s\n", nombre);  **try** {  DataOutputStream dos = **new** DataOutputStream(**new** FileOutputStream(nombre));  long recibidos = 0;  long tam = dis.readLong();  int n;  **while** (recibidos < tam) {  byte[] buffer = **new** byte[1500];  n = dis.read(buffer);  dos.write(buffer, 0, n);  dos.flush();  recibidos += n;  }  dos.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

Esta clase recibirá los archivos y se encargará de crear las carpetas necesarias para su almacenamiento. Es similar a la clase que se encarga del envío; pero a la inversa.

**Clase ListTransferHandler.java**

|  |
| --- |
| **package** view;  **import** **sockets.SocketEnvio**;  **import** **javax.swing.\***;  **import** **java.awt.datatransfer.DataFlavor**;  **import** **java.awt.datatransfer.UnsupportedFlavorException**;  **import** **java.io.File**;  **import** **java.io.IOException**;  **import** **java.util.List**;  **public** **class** **ListTransferHandler** **extends** TransferHandler {  **private** **final** SocketEnvio socketEnvio; *// Socket que usamos en la transferencia*  **private** int action;  ListTransferHandler(int action, SocketEnvio socketEnvio) {  **this**.action = action;  **this**.socketEnvio = socketEnvio;  }  @Override  **public** boolean canImport(TransferHandler.TransferSupport support) {  *// Con esto solo se podran arrastar elementos*  **if** (!support.isDrop()) {  **return** **false**;  }  *// Para solo poder arrastrar archivos/carpetas*  **if** (!support.isDataFlavorSupported(DataFlavor.javaFileListFlavor)) {  System.out.println("NO ES ARCHIVO NI CARPETA");  **return** **false**;  }  boolean actionSupported = (action & support.getSourceDropActions()) == action;  **if** (actionSupported) {  support.setDropAction(action);  **return** **true**;  }  **return** **false**;  }  @Override  **public** boolean importData(TransferHandler.TransferSupport support) {  *// Si no se puede importar el archivo se termina la accion*  **if** (!canImport(support)) {  System.out.println("No se soporta la informacion");  **return** **false**;  }  *// Obtenemos el componente que utiliza drag and drop*  JList jList = (JList) support.getComponent();  DefaultListModel model = **new** DefaultListModel();  jList.setModel(model);  List<File> archivos = **null**;  **try** {  *// Obtenemos los elmeentos arrastrados*  archivos = (List<File>) support.getTransferable()  .getTransferData(DataFlavor.javaFileListFlavor);  *// Los mandamos a su respectivo metodo para ser enviados*  **for** (File file : archivos) {  model.addElement(file.getName());  model.removeElement(file.getName());  *// Manda las carpetas recursivamente*  **if** (file.isDirectory()) socketEnvio.enviarCarpetas(file, "");  **else** socketEnvio.enviarArchivo(file, ""); *// Manda un solo archivo*  model.removeElement(file.getName());  model.addElement(file.getName() + "(enviado)");  }  } **catch** (UnsupportedFlavorException | IOException e) {  e.printStackTrace();  }  **return** **true**;  }  } |

Clase que maneja el drag and drop de la aplicación, para que funcione se tiene que pasar una instancia de esta al elemento que tendrá esta funcionalidad. Su funcionamiento se base en verificar que la acción que se trata de hacer sea válida, validar el tipo de elemento que se está arrastrando y finalmente enviarlo utilizando la clase de envío.

**Métodos importantes de la clase Ventana.java**

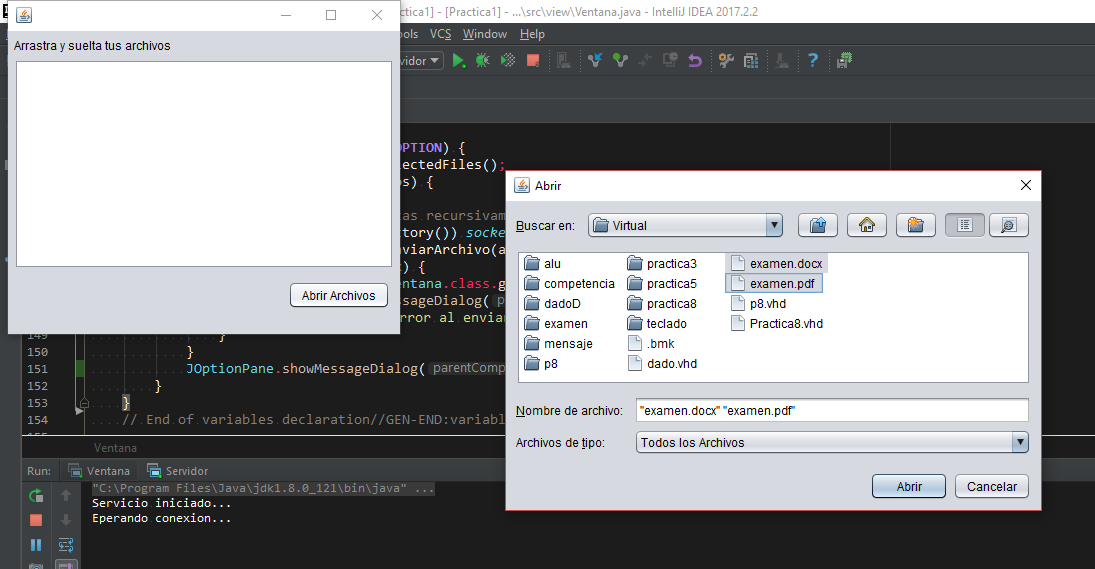
|  |
| --- |
| **private** void btnChooserActionPerformed(java.awt.event.ActionEvent evt) {  *// file chooser que acepta multiples archivos y carpetas*  JFileChooser jf = **new** JFileChooser();  jf.setFileSelectionMode(JFileChooser.FILES\_AND\_DIRECTORIES);  jf.setMultiSelectionEnabled(**true**);  int r = jf.showOpenDialog(**this**);  **if** (r == JFileChooser.APPROVE\_OPTION) {  File archivos[] = jf.getSelectedFiles();  **for** (File archivo : archivos) {  **try** {  *// Manda las carpetas recursivamente*  **if** (archivo.isDirectory()) socketEnvio.enviarCarpetas(archivo, "");  **else** socketEnvio.enviarArchivo(archivo, ""); *// Manda un solo archivo*  } **catch** (IOException ex) {  Logger.getLogger(Ventana.class.getName()).log(Level.SEVERE, **null**, ex);  JOptionPane.showMessageDialog(**this**,  "Error al enviar archivos", "Error", JOptionPane.ERROR\_MESSAGE);  }  }  JOptionPane.showMessageDialog(**this**, "Archivo(s) enviado(s)");  }  }  **private** void myInitComponents() {  *// Creamos nuestro socket de envio*  socketEnvio = **new** SocketEnvio(HOST, PORT);  *// Utilizamos drag an drop sobre la jList*  jListFiles.setTransferHandler(**new** ListTransferHandler(TransferHandler.COPY, socketEnvio));  *// la posición en la que se inserte sera la que ocupe en la jList*  jListFiles.setDropMode(DropMode.INSERT);  } |

Debido a que esta clase tiene mucho código de sobra que se encarga de crear la interfaz de la aplicación solo se muestran estos métodos en donde el primero activa un file chooser para escoger todos los elementos que se vallan a enviar, para hacer esto se establecen los paramentos necesarios. El siguiente método se utiliza para inicializar las variables que se encargan de enviar los archivos y de especificar la función de drag and drop a una lista que utilizamos.

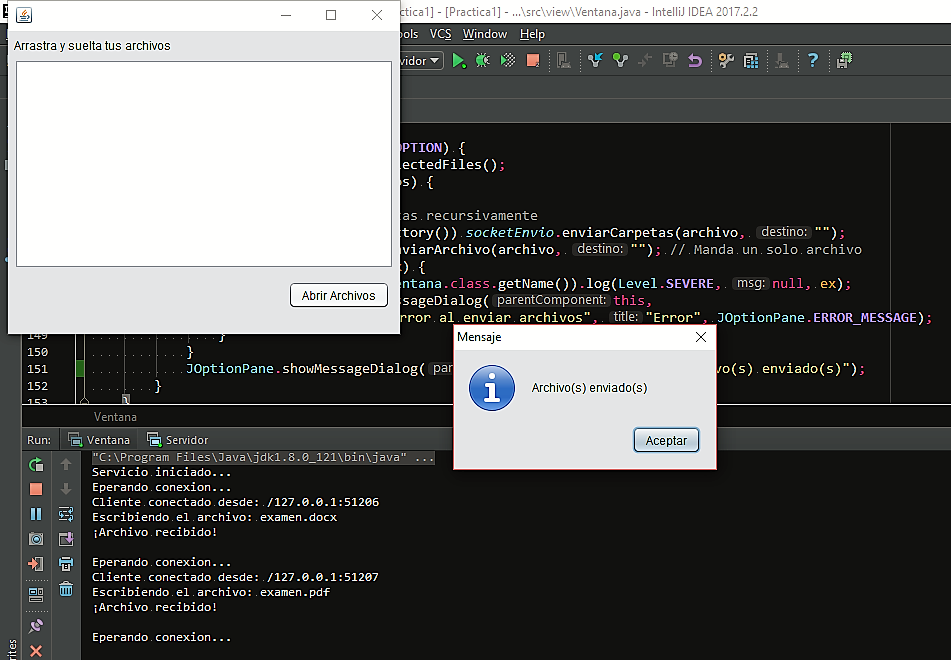
Pruebas

Las pruebas que se realizaron en esta práctica fueron bastante sencillas, básicamente se trata de seleccionar archivos a enviar y después verificar que se hayan creado de manera satisfactoria en el directorio del servidor, tal y como se encontraban en el cliente.

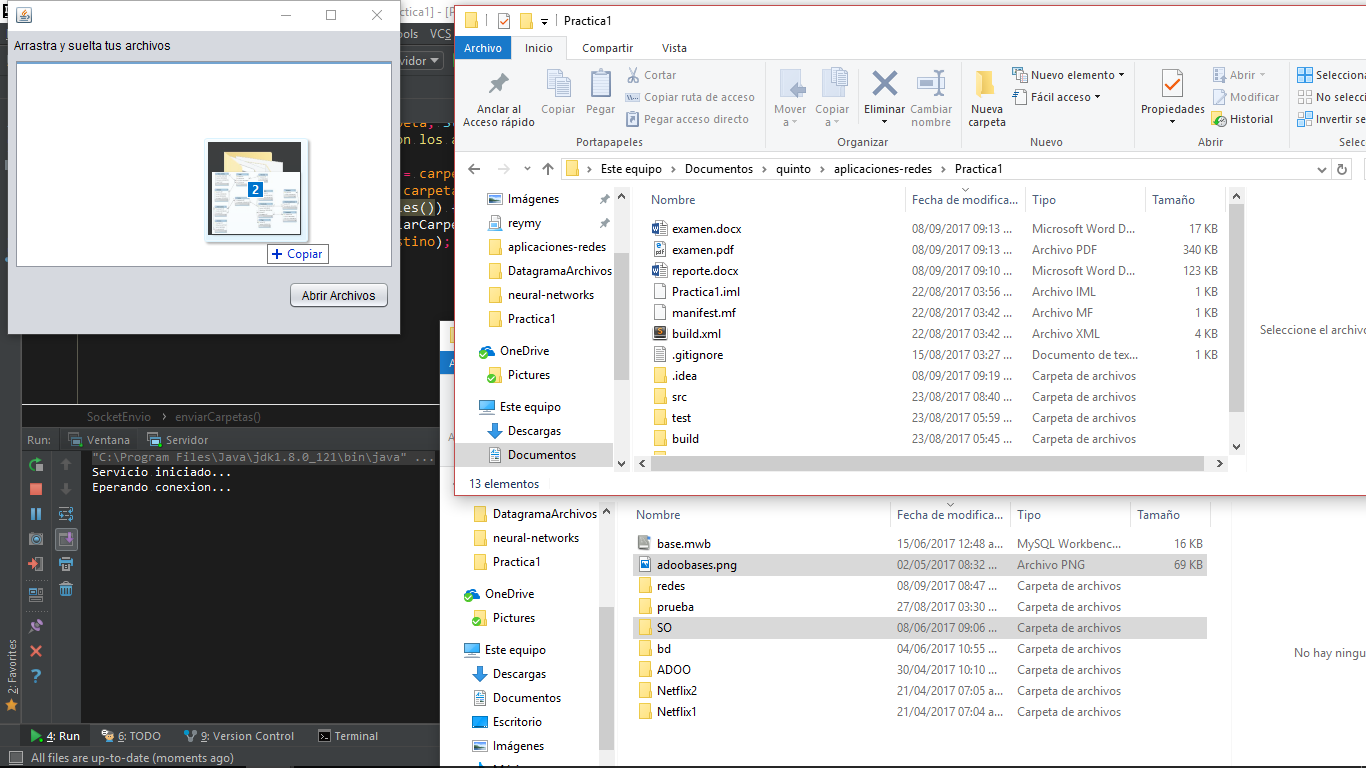
Primero seleccionamos archivos con el file chooser.



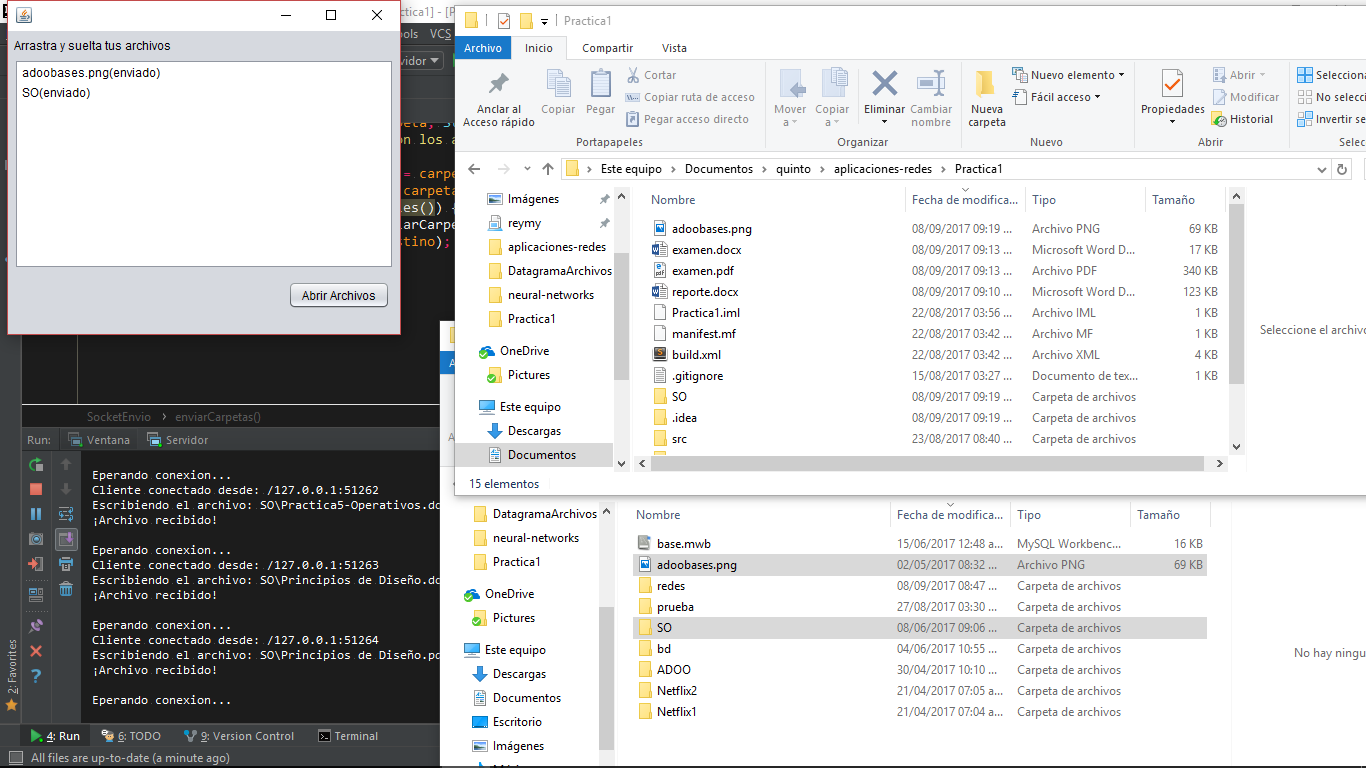
Al hacer esto se envían los archivos, esto se puede apreciar en la consola.



La segunda prueba que se realizo fue arrastrando una carpeta y una imagen, en la imagen se puede observar que estos archivos no existen en el directorio destino.



Como se puede observar en la imagen final se crean los archivos y directorios de esta segunda prueba y la anterior.



Conclusiones

Esta práctica solo es una pequeña muestra de la utilidad que tienen lo sockets en el desarrollo de aplicaciones. En este caso, el envio y almacenamiento de archivos es un caso práctico que muchos servicios de internet suelen usar por lo que el conocer a groso modo una implementación de este tipo nos brinda experiencia que nos puede servir en un futuro.

Además, el hecho de usar sockets de flujo nos facilita el trabajo de envío de archivos ya que es fácil evitar problemas como la perdida de información o el hecho de que los datos lleguen en desorden como ocurriría si se utilizaran sockets de datagrama.

Referencias

* [1] K. Calvert and M. Donahoo, TCP/IP sockets in Java. San Francisco: Morgan Kaufmann Publishers, 2002.