|  |
| --- |
| **Transmisión de video mediante Cliente-Servidor** |
| Redes |
|  |
|  |
|  |
| **Raúl Pérula Martínez** |
| **2011** |
| [88x31.png](http://creativecommons.org/licenses/by-sa/4.0/)  This work by [Raúl Pérula-Martínez](http://educatech.sytes.net/raul/) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-sa/4.0/). |

Índice

[Índice de ilustraciones II](#_Toc294553599)

[1. Introducción 1](#_Toc294553600)

[1.1. Protocolo de Transporte en Tiempo Real (RTP) 1](#_Toc294553601)

[1.1.1. Funcionamiento 1](#_Toc294553602)

[1.1.2. Encabezado RTP 1](#_Toc294553603)

[1.1.3. Aspectos de seguridad 3](#_Toc294553604)

[1.2. Protocolo de Flujo de Video en Tiempo Real (RTSP) 3](#_Toc294553605)

[1.2.1. Funcionamiento 4](#_Toc294553606)

[2. Objetivos 6](#_Toc294553607)

[3. Recursos 7](#_Toc294553608)

[3.1. Recursos humanos 7](#_Toc294553609)

[3.2. Recursos materiales 7](#_Toc294553610)

[3.2.1. Recursos hardware 7](#_Toc294553611)

[3.2.2. Recursos software 7](#_Toc294553612)

[4. Diseño 8](#_Toc294553613)

[4.1. Diagrama de clases 8](#_Toc294553614)

[4.2. Código fuente para la versión 1 8](#_Toc294553615)

[4.2.1. Servidor.java 8](#_Toc294553616)

[4.2.2. Cliente.java 15](#_Toc294553617)

[4.2.3. StreamVideo.java 25](#_Toc294553618)

[4.2.4. PaqueteRTP.java 26](#_Toc294553619)

[4.3. Código fuente para la versión 2 30](#_Toc294553620)

[4.3.1. Servidor.java 30](#_Toc294553621)

[4.3.2. Cliente.java 38](#_Toc294553622)

[4.3.3. VideoStream.java 49](#_Toc294553623)

[4.3.4. PaqueteRTP.java 49](#_Toc294553624)

[5. Futuras mejoras 50](#_Toc294553625)

[Bibliografía 51](#_Toc294553626)

[Apéndice A. Instalación y desinstalación 52](#_Toc294553627)

[Apéndice B. Manual de usuario 53](#_Toc294553628)

# Índice de ilustraciones

[Ilustración 1. Interfaz Servidor( v1). 8](#_Toc294553629)

[Ilustración 2. Interfaz Cliente (v1). 16](#_Toc294553630)

[Ilustración 3. Interfaz Servidor (v2). 30](#_Toc294553631)

[Ilustración 4. Interfaz Cliente (v2). 39](#_Toc294553632)

[Ilustración 5. Servidor. Momento inicial (v1). 53](#_Toc294553633)

[Ilustración 6. Servidor. En reproducción (v1). 53](#_Toc294553634)

[Ilustración 7. Servidor. Momento inicial (v2). 53](#_Toc294553635)

[Ilustración 8. Servidor. Preparado para reproducción (v2). 54](#_Toc294553636)

[Ilustración 9. Servidor. En reproducción (v2). 54](#_Toc294553637)

[Ilustración 10. Cliente. Preparado para pulsar Setup (v1). 55](#_Toc294553638)

[Ilustración 11. Cliente. En reproducción (v1). 55](#_Toc294553639)

[Ilustración 12. Cliente. Preparado para pulsar sobre Conectar (v2). 56](#_Toc294553640)

[Ilustración 13. Cliente. En reproducción (v2). 57](#_Toc294553641)

# Introducción

## Protocolo de Transporte en Tiempo Real (RTP)

El protocolo RTP (Real-time Transport Protocol), que en español es Protocolo de Transporte en tiempo real surgió con la idea de crear un protocolo específico para la gran demanda de recursos en tiempo real por parte de los usuarios. Algunos de estos recursos son la música, videoconferencia, video, telefonía en Internet y más aplicaciones multimedia.

Está formado conjuntamente con el protocolo RTCP (RTP Control Protocol), es decir, Protocolo de Control RTP, cuya función principal es proporcionar mecanismos de realimentación para informar sobre la calidad en la distribución de los datos.

En 1996 se publica en el **RFC** **1889** el estándar del protocolo RTP.

### Funcionamiento

El protocolo RTP se establece en el espacio de usuario y se ejecuta, por lo general, sobre UDP, ya que posee menor retardo que TCP. Por tanto con UDP se gana velocidad a cambio de sacrificar la confiabilidad que TCP ofrece. Debido a ésto, RTP no garantiza la entrega de todos los paquetes, ni la llegada de éstos en el instante adecuado.

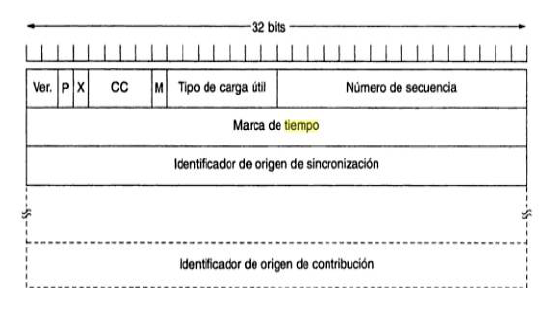
La función básica de RTP es multiplexar varios flujos de datos en tiempo real en un solo flujo de paquetes UDP, pudiéndose enviar tanto a un solo destino (unicast) o múltiples destinos (multicast). Los paquetes son numerados de la siguiente manera: se le asigna a cada paquete un número mayor que su antecesor. Esto será útil para que la aplicación conozca si ha fallado algún paquete o no en la transmisión. Si ha fallado, al no tener un control de flujo, de errores, de confirmaciones de recepción ni de solicitud de transmisión, la mejor opción es la interpolación de los datos.

Otra característica muy importante para las aplicaciones de contenido multimedia en tiempo real es el time-stamping (marcación del tiempo).

La idea es permitir que el origen asocie una marca de tiempo con la primera muestra de cada paquete. Las marcas de tiempo son relativas al inicio del flujo, por tanto, solo importa las diferencias entre dichas marcas de tiempo. Con este planteamiento, el destino es capaz de almacenar un pequeño buffer e ir reproduciendo cada muestra el número exacto de milisegundos después del inicio del flujo reduciendo los efectos de la fluctuación y sincronizando múltiples flujos entre sí.

### Encabezado RTP

El encabezado de los paquetes RTP consiste en 3 palabras de 32 bits y algunas extensiones:



**PRIMERA PALABRA:**

* ***Versión***: campo versión (2 bits).
* ***P***: indica si el paquete se ha rellenado a un múltiplo de 4 bytes. El último byte de relleno indica cuántos bytes se agregaron (1 bit).
* ***X***: indica si hay un encabezado de extensión (1 bit).
* ***CC***: indica cuántos orígenes de contribución están presentes, de 0 a 15 (4 bits).
* ***M***: es un marcador específico de la aplicación, normalmente un marcador de inicio (1 bit).
* ***Tipo de carga útil***: indica cuál es el algoritmo de codificación que se ha utilizado (7 bits).
* ***Número de secuencia***: contador que se incrementa en cada paquete RTP enviado (16 bits).

**SEGUNDA PALABRA:**

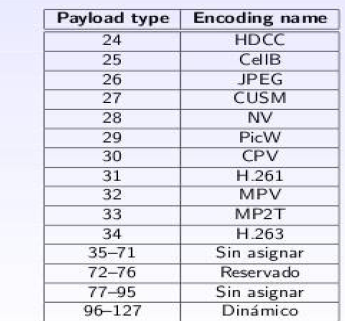
* ***Marca de tiempo***: indica cuándo se creó la primera muestra en el paquete (32 bits).

**TERCERA PALABRA:**

* ***Identificador de origen de sincronización***: indica a cuál flujo pertenece el paquete. Es el método para de multiplexar/demultiplexar varios flujos de datos en un solo flujo de paquetes UDP (32 bits).
* Por último, los ***Identificadores de origen de contribución***, en caso de que haya, se utilizan cuando los mezcladores están presentes en el estudio. En ese caso, el mezclador es el origen de sincronización, y los flujos que se mezclan se listan en esta palabra.

**TIPO DE CARGA ÚTIL (Payload)**

A continuación se muestran dos tablas con los posibles tipos de carga útil, es decir, el algoritmo de codificación asociado.



### Aspectos de seguridad

RTP sufre vulnerabilidades al igual que otros protocolos. Por ejemplo, un usuario atacante podría autenticar de forma falsa direcciones de red de origen o destino, cambiar el encabezado e incluso cambiar el algoritmo de codificación.

Utilizando el protocolo RTP sin su protocolo de control RTCP, los campos CNAME y NAME podrían usarse para autenticar a otro usuario. Debido a estas vulnerabilidades entre otras, es importante saber unos cuantos aspectos de seguridad para hacer un uso más responsable del protocolo.

RTP es usado actualmente en la telefonía VoIP, llamadas telefónicas a través de Internet. Por tanto, la captura de paquetes RTP es un problema para la integridad de la conversación debido a las vulnerabilidades en seguridad. El tema de vulnerabilidades y agujeros en seguridad está siendo un tema de actualidad debido a los problemas que plantean para los usuarios.

## Protocolo de Flujo de Video en Tiempo Real (RTSP)

El protocolo RTSP (Real-Time Streaming Protocol), es un protocolo basado en texto e independiente del protocolo de transporte que permite realizar un control remoto de sesión de transmisión multimedia que permite:

* Recuperar un determinado medio de un servidor.
* Invitar a un servidor de medios a una multiconferencia.
* Grabar una multiconferencia.

En 1998 se publica en el **RFC 2326** el estándar del protocolo RTP.

### Funcionamiento

El protocolo recuerda en diseño, en parte, a HTTP. Emplea URLs para la transmisión.

Se manda un mensaje se solicitud a una URL en un paquete TCP.

|  |
| --- |
| PLAY rtsp://video.example.com/conf1/video1 RTSP/1.0  CSeq: 2  Session: 123456  Range: smpte=0:10:00- |

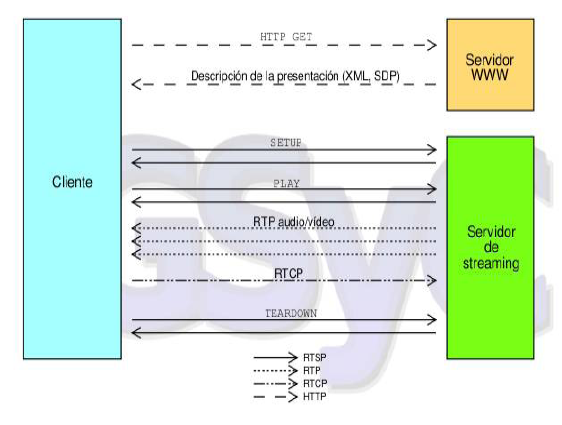
El servidor de medios dará una respuesta en un paquete TCP.

|  |
| --- |
| RTSP/1.0 200 OK  CSeq: 2  Session: 1234567  Range: smpte=0:10:00-0:20:00  RTP-Info: url=rtsp://video.example.com/conf1/video1;  seq=123123123;rtptime=456456456 |

Los mensajes principales de solicitud que se pueden mandar pueden ser del tipo:

* ***SETUP***: El servidor asigna recursos y establece una sesión RTSP.
* ***PLAY***: Empieza la transmisión de datos.
* ***PAUSE***: Detiene temporalmente la transmisión.
* ***TEARDOWN***: Libera los recursos y termina la sesión RTSP.

El esquema de funcionamiento de RTSP es el siguiente:



# Objetivos

El objetivo principal de este trabajo es el estudio y comprensión de los protocolos de transmisión de vídeo junto con el manejo de un lenguaje como es JAVA para la utilización de transmisión de vídeo en streaming haciendo uso de un modelo de los más conocidos en Redes como es el modelo Cliente/Servidor.

Como objetivos secundarios se encuentran, entre otros, la investigación de la captación de vídeo desde la webcam para el posterior uso del vídeo y envío del mismo. Intentando conseguir una retransmisión del mismo en tiempo real para la posterior realización de un sistema de multi-vídeoconferencias a partir del modelo usado.

Todo esto se realizará con las implementaciones de los protocolos RTP y RTSP para la transmisión de vídeo, explicados en los puntos anteriores.

# Recursos

Para la realización del proyecto se dispondrán de los siguientes recursos:

## Recursos humanos

**Autor**: Raúl Pérula Martínez.

*Alumno de 2º de Ingeniería Informática.*

**Profesor**: D. Enrique García Salcines.

*Profesor Contratado Doctor.*

## Recursos materiales

### Recursos hardware

Equipo portátil con las siguientes características técnicas:

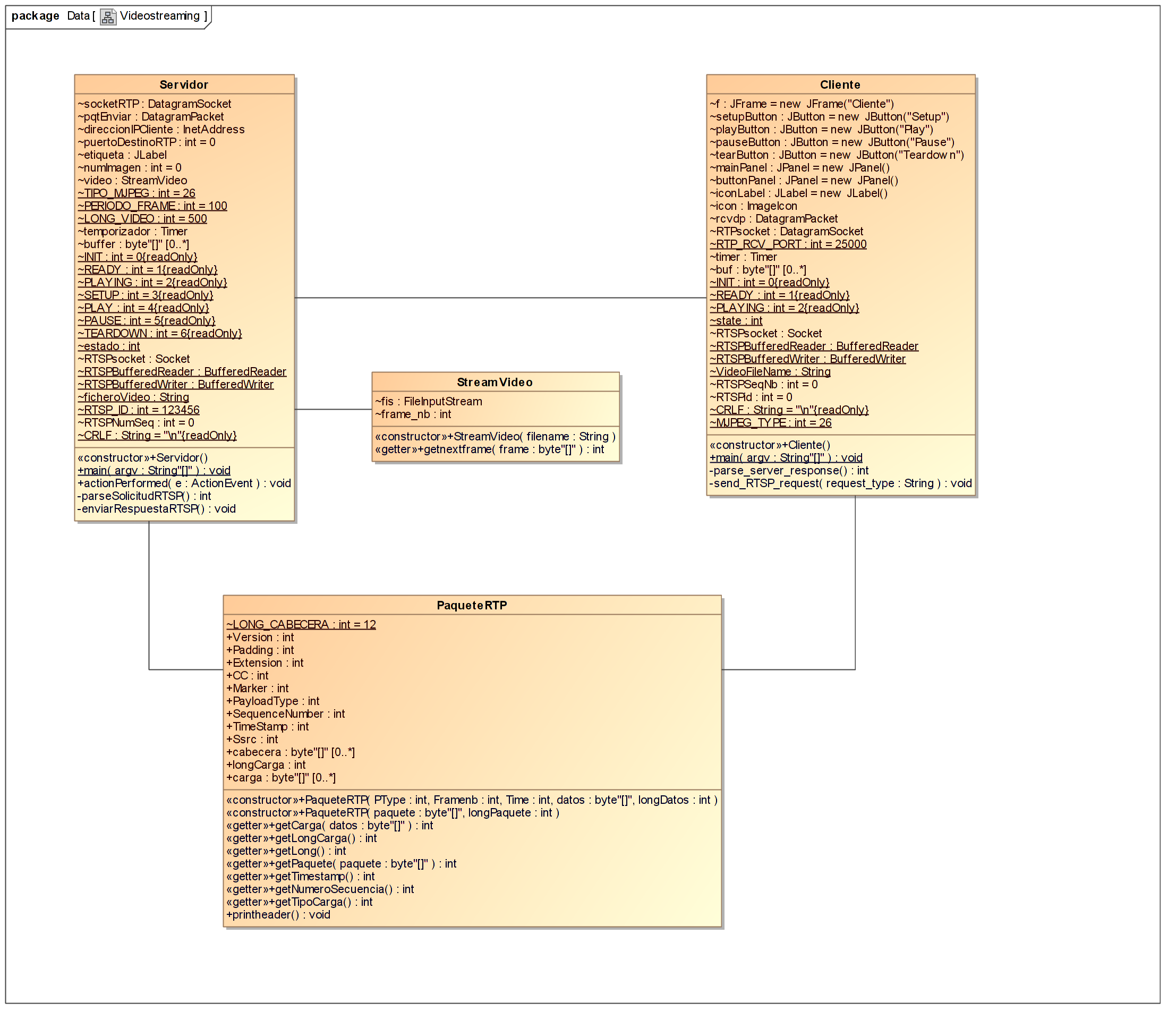
* Procesador Intel Core 2 Duo T7300 (Santa Rosa) a 2 GHz (4 MB memoria caché L2).
* 2 GB memoría RAM.
* Disco duro 120 GB.

### Recursos software

* Sistema operativo Ubuntu 10.04.
* Eclipse Java EE IDE for Web Developers. Version: Helios Service Release 1.
* JMF 2.1.1.
* Java versión 1.6.0\_20.

# Diseño

## Diagrama de clases



## Código fuente para la versión 1

Esta primera versión contiene la implementación básica del Cliente/Servidor de videostreaming. La interfaz es la más simple posible ya que se ha centrado más en darle importancia a la comunicación y a la estructuración del envío y recepción de los paquetes RTP para la transmisión del vídeo.

### Servidor.java

Servidor_001.png

Ilustración . Interfaz Servidor( v1).

|  |
| --- |
| import java.io.\*;  import java.net.\*;  import java.awt.\*;  import java.util.\*;  import java.awt.event.\*;  import javax.swing.\*;  import javax.swing.Timer;  /\*\*  \* @author Raul Perula Martinez <i62pemar@uco.es>  \* @version 1.0  \*  \* Clase del servidor que implementa la ventana grafica junto con las  \* acciones del envio del video a traves de streaming a un cliente.  \*/  public class Servidor extends JFrame implements ActionListener {  /\*\*  \* Variables RTP.  \*/  DatagramSocket socketRTP; // Socket para enviar y recibir datagramas RTP  DatagramPacket pqtEnviar; // Paquete UDP con los frames de video  InetAddress direccionIPCliente; // Direccion IP del Cliente  int puertoDestinoRTP = 0; // Puerto destino para los paquetes  // RTP (proporcionado por el cliente RTSP)  /\*\*  \* GUI.  \*/  JLabel etiqueta;  /\*\*  \* Variable de video.  \*/  int numImagen = 0; // Numero de imagen transmitida en este momento  StreamVideo video; // VideoStream utilizado para acceder a los frames de  // video  static int TIPO\_MJPEG = 26; // Tipo de carga RTP para video MJPEG  static int PERIODO\_FRAME = 100; // Periodo de frames a transmitir, en  // milisegundos  static int LONG\_VIDEO = 500; // Longitud del video en frames  Timer temporizador; // Temporizador utilizado para enviar las imagenes del  // video al ratio adecuado  byte[] buffer; // Buffer utilizado para almacenar las imagenes a enviar al  // cliente  /\*\*  \* Variables RTSP. Estados RTSP.  \*/  final static int INIT = 0;  final static int READY = 1;  final static int PLAYING = 2;  /\*\*  \* Tipos de mensajes RTSP.  \*/  final static int SETUP = 3;  final static int PLAY = 4;  final static int PAUSE = 5;  final static int TEARDOWN = 6;  static int estado; // Estado del servidor RTSP == INIT o READY o PLAY  Socket RTSPsocket; // Socket utilizado para recibir y enviar mensajes RTSP  /\*\*  \* Canales de entrada y salida.  \*/  static BufferedReader RTSPBufferedReader;  static BufferedWriter RTSPBufferedWriter;  static String ficheroVideo; // Fichero de video solicitado por el cliente  static int RTSP\_ID = 123456; // ID of the RTSP session  int RTSPNumSeq = 0; // Sequence number of RTSP messages within the session  final static String CRLF = "\n";  /\*\*  \* Constructor.  \*/  public Servidor() {  /\*\*  \* Inicialiar Frame.  \*/  super("Servidor");  /\*\*  \* Inicializar temporizador.  \*/  temporizador = new Timer(PERIODO\_FRAME, this);  temporizador.setInitialDelay(0);  temporizador.setCoalesce(true);  /\*\*  \* Reservar memoria para el buffer de envio.  \*/  buffer = new byte[15000];  /\*\*  \* Manejador para cerrar la ventana principal.  \*/  addWindowListener(new WindowAdapter() {  public void windowClosing(WindowEvent e) {  /\*\*  \* Parar el temporizador y salir.  \*/  temporizador.stop();  System.exit(0);  }  });  /\*\*  \* GUI.  \*/  etiqueta = new JLabel("Enviando frame # ", JLabel.CENTER);  getContentPane().add(etiqueta, BorderLayout.CENTER);  }  /\*\*  \* Funcion principal.  \*  \* @param argv  \* Entrada por linea de comandos.  \* @throws Exception  \*/  public static void main(String argv[]) throws Exception {  if (argv.length != 1) {  System.out.println("Utilizacion: java Servidor [puerto RTSP]");  System.exit(0);  }  /\*\*  \* Crear objeto Servidor.  \*/  Servidor servidor = new Servidor();  /\*\*  \* Mostrar GUI.  \*/  servidor.pack();  servidor.setVisible(true);  /\*\*  \* Obtener el puerto RTSP.  \*/  int RTSPport = Integer.parseInt(argv[0]);  /\*\*  \* Iniciar la conexion TCP con el cliente para la sesion RTSP.  \*/  ServerSocket socketServidor = new ServerSocket(RTSPport);  servidor.RTSPsocket = socketServidor.accept();  socketServidor.close();  /\*\*  \* Obtener direccion IP del cliente.  \*/  servidor.direccionIPCliente = servidor.RTSPsocket.getInetAddress();  /\*\*  \* Inicializar el estado RTSP.  \*/  estado = INIT;  /\*\*  \* Establecer los canales de entrada y salida.  \*/  RTSPBufferedReader = new BufferedReader(new InputStreamReader(  servidor.RTSPsocket.getInputStream()));  RTSPBufferedWriter = new BufferedWriter(new OutputStreamWriter(  servidor.RTSPsocket.getOutputStream()));  /\*\*  \* Esperar por el mensaje SETUP del cliente.  \*/  int tipoSolicitud;  boolean acabo = false;  while (!acabo) {  tipoSolicitud = servidor.parseSolicitudRTSP(); // Bloqueante  if (tipoSolicitud == SETUP) {  acabo = true;  /\*\*  \* Actualizar estado RTSP.  \*/  estado = READY;  System.out.println("Nuevo estado RTSP: READY");  /\*\*  \* Enviar respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Inicializar el objeto VideoStream.  \*/  servidor.video = new StreamVideo(ficheroVideo);  /\*\*  \* Inicializar el socket RTP.  \*/  servidor.socketRTP = new DatagramSocket();  }  }  /\*\*  \* Bucle para gestionar las solicitudes RTSP.  \*/  while (true) {  /\*\*  \* Analizar la solicitud.  \*/  tipoSolicitud = servidor.parseSolicitudRTSP(); // Bloqueante  if ((tipoSolicitud == PLAY) && (estado == READY)) {  /\*\*  \* Enviar la respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Iniciar temporizador.  \*/  servidor.temporizador.start();  /\*\*  \* Actualizar estado.  \*/  estado = PLAYING;  System.out.println("Nuevo estado RTSP: PLAYING");  } else if ((tipoSolicitud == PAUSE) && (estado == PLAYING)) {  /\*\*  \* Enviar la respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Parar temporizador.  \*/  servidor.temporizador.stop();  /\*\*  \* Actualizar estado.  \*/  estado = READY;  System.out.println("Nuevo estado RTSP: READY");  } else if (tipoSolicitud == TEARDOWN) {  /\*\*  \* Enviar la respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Parar temporizador.  \*/  servidor.temporizador.stop();  /\*\*  \* Cerrar sockets.  \*/  servidor.RTSPsocket.close();  servidor.socketRTP.close();  System.exit(0);  }  }  }  /\*  \* (non-Javadoc) Manejador del temporizador  \*  \* @see  \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)  \*/  public void actionPerformed(ActionEvent e) {  /\*\*  \* Si el numero de imagen es menor que la longitud del video.  \*/  if (numImagen < LONG\_VIDEO) {  /\*\*  \* Incrementar numImagen.  \*/  numImagen++;  try {  /\*\*  \* Obtener el siguiente frame y su tamaño.  \*/  int longImagen = video.getnextframe(buffer);  /\*\*  \* Construir un paquete RTP con el frame.  \*/  PaqueteRTP paqueteRTP = new PaqueteRTP(TIPO\_MJPEG, numImagen,  numImagen \* PERIODO\_FRAME, buffer, longImagen);  /\*\*  \* Obtener la longitud total del paquete RTP a enviar.  \*/  int longPaquete = paqueteRTP.getLong();  /\*\*  \* Recuperar el array de bits del paquete y almacenarlo en un  \* array.  \*/  byte[] bitsPaquete = new byte[longPaquete];  paqueteRTP.getPaquete(bitsPaquete);  /\*\*  \* Enviar el paquete como un DatagramPacket sobre un socket UDP.  \*/  pqtEnviar = new DatagramPacket(bitsPaquete, longPaquete,  direccionIPCliente, puertoDestinoRTP);  socketRTP.send(pqtEnviar);  System.out.println("Enviando frame #" + numImagen);  /\*\*  \* Mostrar la cabecera.  \*/  paqueteRTP.printheader();  /\*\*  \* Actualizar GUI.  \*/  etiqueta.setText("Enviando frame #" + numImagen);  } catch (Exception ex) {  System.out.println("Excepcion 1: " + ex);  System.exit(0);  }  } else {  /\*\*  \* Si se ha alcanzado el final del video, parar temporizador  \*/  temporizador.stop();  }  }  /\*\*  \* Analizar solicitud RTSP.  \*  \* @return Devuelve  \*/  private int parseSolicitudRTSP() {  int tipoSolicitud = -1;  try {  /\*\*  \* Parsear la linea de solicitud y extraer el tipo de solicitud.  \*/  String lineaSolicitud = RTSPBufferedReader.readLine();  // System.out.println("Servidor RTSP - Recibido del cliente:");  System.out.println(lineaSolicitud);  StringTokenizer tokens = new StringTokenizer(lineaSolicitud);  String stringTipoSolicitud = tokens.nextToken();  /\*\*  \* Convertir a una estructura tipoSolicitud.  \*/  if ((new String(stringTipoSolicitud)).compareTo("SETUP") == 0)  tipoSolicitud = SETUP;  else if ((new String(stringTipoSolicitud)).compareTo("PLAY") == 0)  tipoSolicitud = PLAY;  else if ((new String(stringTipoSolicitud)).compareTo("PAUSE") == 0)  tipoSolicitud = PAUSE;  else if ((new String(stringTipoSolicitud)).compareTo("TEARDOWN") == 0)  tipoSolicitud = TEARDOWN;  if (tipoSolicitud == SETUP) {  /\*\*  \* Extraer el fichero de video de la linea de solicitud.  \*/  ficheroVideo = tokens.nextToken();  }  /\*\*  \* Parsear la linea de numero de secuencia.  \*/  String lineaNumSeq = RTSPBufferedReader.readLine();  System.out.println(lineaNumSeq);  tokens = new StringTokenizer(lineaNumSeq);  tokens.nextToken();  RTSPNumSeq = Integer.parseInt(tokens.nextToken());  /\*\*  \* Obtener la ultima linea.  \*/  String ultimaLinea = RTSPBufferedReader.readLine();  System.out.println(ultimaLinea);  if (tipoSolicitud == SETUP) {  /\*\*  \* Extraer el puerto destino RTP de la ultima linea.  \*/  tokens = new StringTokenizer(ultimaLinea);  for (int i = 0; i < 3; i++)  tokens.nextToken(); // no utilizado  puertoDestinoRTP = Integer.parseInt(tokens.nextToken());  }  /\*\*  \* Sino la ultima linea es la linea de identificador de sesion, no  \* se analiza por ahora.  \*/  } catch (Exception ex) {  System.out.println("Excepcion 2: " + ex);  System.exit(0);  }  return (tipoSolicitud);  }  /\*\*  \* Enviar respuesta RTSP.  \*/  private void enviarRespuestaRTSP() {  try {  RTSPBufferedWriter.write("RTSP/1.0 200 OK" + CRLF);  RTSPBufferedWriter.write("CSeq: " + RTSPNumSeq + CRLF);  RTSPBufferedWriter.write("Session: " + RTSP\_ID + CRLF);  RTSPBufferedWriter.flush();  // System.out.println("Servidor RTSP - Enviando respuesta al cliente.");  } catch (Exception ex) {  System.out.println("Excepcion 3: " + ex);  System.exit(0);  }  }  } |

### Cliente.java

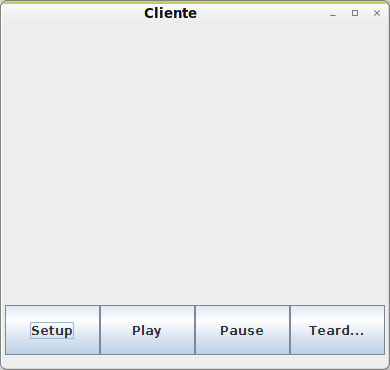


Ilustración . Interfaz Cliente (v1).

|  |
| --- |
| import java.io.\*;import java.net.\*;import java.util.\*;import java.awt.\*;import java.awt.event.\*;import javax.swing.\*;import javax.swing.Timer;  /\*\* \* @author Raul Perula Martinez <i62pemar@uco.es> \* @version 1.0 \* \* Clase del cliente que implementa los metodos necesarios para \* conectarse con el servidor, reproducir, pausar el video enviado en \* streaming y finalizar desconectandose de la conexion. \*/  public class Cliente {  /\*\* \* GUI. \*/  JFrame f = new JFrame("Cliente"); JButton setupButton = new JButton("Setup"); JButton playButton = new JButton("Play"); JButton pauseButton = new JButton("Pause"); JButton tearButton = new JButton("Teardown");  JPanel mainPanel = new JPanel(); JPanel buttonPanel = new JPanel(); JLabel iconLabel = new JLabel();  ImageIcon icon;  /\*\* \* RTP variables. \*/  DatagramPacket rcvdp; // Paquete UDP recibido desde el servidor DatagramSocket RTPsocket; // Socket usado para enviar y recibir paquetes UDP  static int RTP\_RCV\_PORT = 25000; // Puerto por donde el cliente recibira los  // paquetes RTP  Timer timer; // Temporizador usado para recibir datos desde el socket UDP  byte[] buf; // Bufer usado para almacenar datos recibidos desde el servidor  /\*\* \* Variables RTSP. Estados RTSP. \*/ final static int INIT = 0; final static int READY = 1; final static int PLAYING = 2; static int state; // Estado RTSP == INIT o READY o PLAYING Socket RTSPsocket; // Socket usado para enviar/recibir mensajes RTSP  /\*\* \* Flujos de entrada y salida. \*/ static BufferedReader RTSPBufferedReader; static BufferedWriter RTSPBufferedWriter; static String VideoFileName; // Fichreo de video para pedir al servidor int RTSPSeqNb = 0; // Numero de secuencia de los mensajes RTSP sin la sesion int RTSPid = 0; // ID de la sesion RTSP (dada por el servidor RTSP)  final static String CRLF = "\n";  /\*\* \* Constantes de video. \*/ static int MJPEG\_TYPE = 26; // Tipo de RTP para video MJPEG  /\*\* \* Constructor. \*/ public Cliente() { /\*\* \* Construccion de la GUI. \*/  /\*\* \* Frame. \*/ f.addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent e) { System.exit(0); } }); /\*\* \* Botones. \*/  buttonPanel.setLayout(new GridLayout(1, 0)); buttonPanel.add(setupButton); buttonPanel.add(playButton); buttonPanel.add(pauseButton); buttonPanel.add(tearButton);  setupButton.addActionListener(new setupButtonListener()); playButton.addActionListener(new playButtonListener()); pauseButton.addActionListener(new pauseButtonListener()); tearButton.addActionListener(new tearButtonListener());  /\*\* \* Etiqueta de imagen mostrada. \*/ iconLabel.setIcon(null);  /\*\* \* Capa de salida del frame. \*/  mainPanel.setLayout(null); mainPanel.add(iconLabel); mainPanel.add(buttonPanel); iconLabel.setBounds(0, 0, 380, 280); buttonPanel.setBounds(0, 280, 380, 50);  f.getContentPane().add(mainPanel, BorderLayout.CENTER); f.setSize(new Dimension(390, 370)); f.setVisible(true);  /\*\* \* Inicializacion del temporizados. \*/ timer = new Timer(20, new timerListener()); timer.setInitialDelay(0); timer.setCoalesce(true);  /\*\* \* Almacenamiento de suficiente memoria para el bufer usado para recibir \* datos desde el servidor. \*/ buf = new byte[15000]; }  /\*\* \* Funcion principal. \* \* @param argv \* Lista de argumentos en linea de comandos. \* @throws Exception \*/ public static void main(String argv[]) throws Exception { if (argv.length != 3) { System.out .println("Utilizacion: java Cliente [hostname Servidor] [puerto Servidor RTSP] [fichero de video]"); System.exit(0); }  /\*\* \* Creacion de un objeto Cliente. \*/ Cliente theClient = new Cliente();  /\*\* \* Obtener puerto RTSP del servidor y la direccion IP desde linea de \* comandos. \*/ int RTSP\_server\_port = Integer.parseInt(argv[1]); String ServerHost = argv[0]; InetAddress ServerIPAddr = InetAddress.getByName(ServerHost);  /\*\* \* Obtener nombre de fichero de video. \*/ VideoFileName = argv[2];  /\*\* \* Establecer una conexion TCP con el servidor para intercambiar \* mensajes RTSP. \*/ theClient.RTSPsocket = new Socket(ServerIPAddr, RTSP\_server\_port);  /\*\* \* Poner los flujos de entrada y salida. \*/ RTSPBufferedReader = new BufferedReader(new InputStreamReader( theClient.RTSPsocket.getInputStream())); RTSPBufferedWriter = new BufferedWriter(new OutputStreamWriter( theClient.RTSPsocket.getOutputStream()));  /\*\* \* Inicializar estado RTSP. \*/ state = INIT; }  /\*\* \* Funciones de los botones. \*/  /\*\* \* Funcion para el boton Setup. \*/ class setupButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { if (state == INIT) { /\*\* \* Inicializar un RTPsocket no bloqueante que sera usado para \* recibir los datos. \*/ try { /\*\* \* Construir un nuevo DatagramSocket para recibir los \* paquetes RTP desde el servidor, en el puerto \* RTP\_RCV\_PORT. \*/ RTPsocket = new DatagramSocket(RTP\_RCV\_PORT);  /\*\* \* Poner el valor de TimeOut a 5ms. \*/ RTPsocket.setSoTimeout(5); } catch (SocketException se) { System.out.println("Excepcion de socket: " + se); System.exit(0); }  /\*\* \* Inicializar el numero de secuencia RTSP. \*/ RTSPSeqNb = 1;  /\*\* \* Enviar mensaje SETUP al servidor. \*/ send\_RTSP\_request("SETUP");  /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor Invalida"); else { /\*\*  \* Cambiar el estado RTSP e imprimir el nuevo estado. \*/ state = READY; System.out.println("Nuevo estado RTSP: READY"); } } } }  /\*\* \* Funcion para el boton Play. \*/ class playButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { if (state == READY) { /\*\* \* Incremento del numero de secuencia RTSP. \*/ RTSPSeqNb++;  /\*\* \* Enviar mensaje PLAY al servidor. \*/ send\_RTSP\_request("PLAY");  /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor Invalida"); else { /\*\* \* Cambiar el estado RTSP e imprimir el nuevo estado. \*/ state = PLAYING; System.out.println("Nuevo estado RTSP: PLAYING");  /\*\* \* Activar el temporizador. \*/ timer.start(); } } } }  /\*\* \* Funcion del boton Pause. \*/ class pauseButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { if (state == PLAYING) { /\*\* \* Incrementar el numero de secuencia RTSP. \*/ RTSPSeqNb++;  /\*\* \* Enviar mensaje PAUSE al servidor. \*/ send\_RTSP\_request("PAUSE");  /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor Invalida"); else { /\*\* \* Cambiar estado RTSP e imprimir el nuevo estado. \*/ state = READY; System.out.println("Nuevo estado RTSP: READY");  /\*\* \* Parar el temporizador. \*/ timer.stop(); } } } }  /\*\* \* Funcion del boton Teardown. \*/ class tearButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { /\*\* \* Incrementar el numero de secuencia RTSP. \*/ RTSPSeqNb++;  /\*\* \* Enviar mensaje TEARDOWN al servidor. \*/ send\_RTSP\_request("TEARDOWN");  /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor Invalida"); else { /\*\* \* Cambiar estado RTSP e imprimir el nuevo estado. \*/ state = INIT; System.out.println("Nuevo estado RTSP: INIT");  /\*\* \* Parar el temporizador. \*/ timer.stop();  /\*\* \* Salir. \*/ System.exit(0); } } }  /\*\* \* Funcion del temporizador. \*/ class timerListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { /\*\* \* Construir un DatagramPacket para recibir datos desde el socket \* UDP. \*/ rcvdp = new DatagramPacket(buf, buf.length);  try { /\*\* \* Recibir el DP desde el socket. \*/ RTPsocket.receive(rcvdp); /\*\* \* Crear un objeto PaqueteRTP desde el DP. \*/ PaqueteRTP rtp\_packet = new PaqueteRTP(rcvdp.getData(), rcvdp .getLength()); /\*\* \* Imprimir los campos de las cabeceras importantes del paquete \* RTP recibido. \*/ System.out .println("Recibido paquete RTP con Numero Secuencia # " + rtp\_packet.getNumeroSecuencia() + " TimeStamp " + rtp\_packet.getTimestamp() + " ms, de tipo " + rtp\_packet.getTipoCarga()); /\*\* \* Imprimir la cabecera bitstream. \*/ rtp\_packet.printheader();  /\*\* \* Obtener el bitstream desde el objeto PaqueteRTP. \*/ int payload\_length = rtp\_packet.getLongCarga(); byte[] payload = new byte[payload\_length]; rtp\_packet.getCarga(payload);  /\*\* \* Obtener una objeto imagen desde el bitstream. \*/ Toolkit toolkit = Toolkit.getDefaultToolkit(); Image image = toolkit.createImage(payload, 0, payload\_length);  /\*\* \* Mostrar la imagen como un objeto ImageIcon. \*/ icon = new ImageIcon(image); iconLabel.setIcon(icon); } catch (InterruptedIOException iioe) { // System.out.println("Nada que leer"); } catch (IOException ioe) { System.out.println("Excepcion capturada: " + ioe); } } }  /\*\* \* Parsear respuesta del servidor. \* \* @return Devuelve el codigo de respuesta. \*/ private int parse\_server\_response() { int reply\_code = 0;  try { /\*\* \* Parsear estado de la linea y extraer el reply\_code. \*/ String StatusLine = RTSPBufferedReader.readLine(); System.out.println("Cliente RTSP - Recibido desde el servidor:"); System.out.println(StatusLine); StringTokenizer tokens = new StringTokenizer(StatusLine); tokens.nextToken(); // Saltar al siguiente RTSP reply\_code = Integer.parseInt(tokens.nextToken());  /\*\* \* Si el reply\_code esta OK se obtiene e imprime las 2 siguiente \* lineas. \*/ if (reply\_code == 200) { String SeqNumLine = RTSPBufferedReader.readLine(); System.out.println(SeqNumLine);  String SessionLine = RTSPBufferedReader.readLine(); System.out.println(SessionLine);  /\*\* \* Si el estado == INIT obtiene el Id de Sesion desde la \* SessionLine. \*/ tokens = new StringTokenizer(SessionLine); tokens.nextToken(); // Saltar a la siguiente sesion RTSPid = Integer.parseInt(tokens.nextToken()); } } catch (Exception ex) { System.out.println("Excepcion capturada: " + ex); System.exit(0); }  return (reply\_code); }  /\*\* \* Enviar peticion RTSP. \* \* @param request\_type \* Tipo de peticion. \*/ private void send\_RTSP\_request(String request\_type) { try { /\*\* \* Usar el RTSPBufferedWriter para escribir el socket RTSP. Escribir \* la linea de peticion. \*/ RTSPBufferedWriter.write(request\_type + " " + VideoFileName + " RTSP/1.0" + CRLF); /\*\* \* Escribir la linea CSeq. \*/ RTSPBufferedWriter.write("CSeq: " + RTSPSeqNb + CRLF); /\*\* \* Comprobar si el request\_type es igual a "SETUP" y en ese caso \* escribir el Transport: linea de aviso al servidor del puerto \* usado para recibir los paquetes RTP, RTP\_RCV\_PORT. \*/ if ((new String(request\_type)).compareTo("SETUP") == 0) RTSPBufferedWriter.write("Transport: RTP/UDP; client\_port= " + RTP\_RCV\_PORT + CRLF); else RTSPBufferedWriter.write("Session: " + RTSPid + "\n"); RTSPBufferedWriter.flush(); } catch (Exception ex) { System.out.println("Excepcion capturada: " + ex); System.exit(0); } }} |

### StreamVideo.java

|  |
| --- |
| import java.io.\*;  /\*\* \* @author Raul Perula Martinez <i62pemar@uco.es> \* @version 1.0 \* \* Clase que se utiliza para el tratado y transmision del video. \*/  public class StreamVideo { FileInputStream fis; // Fichero de video  int frame\_nb; // Frame nb actual  /\*\* \* Constructor. \* \* @param filename \* Nombre del fichero. \* @throws Exception \*/ public StreamVideo(String filename) throws Exception { /\*\* \* Inicializar variables. \*/ fis = new FileInputStream(filename); frame\_nb = 0; }  /\*\* \* Obtener siguiente frame. \* \* @param frame \* @return Devuelve el siguiente frame como un array de bytes y el tamaño \* del frame. \* @throws Exception \*/ public int getnextframe(byte[] frame) throws Exception { int length = 0; String length\_string; byte[] frame\_length = new byte[5];  /\*\* \* Leer el tamaño del frame actual. \*/ fis.read(frame\_length, 0, 5);  /\*\* \* Transformar frame\_length a entero. \*/ length\_string = new String(frame\_length); length = Integer.parseInt(length\_string);  return (fis.read(frame, 0, length)); }} |

### PaqueteRTP.java

|  |
| --- |
| /\*\* \* @author Raul Perula Martinez <i62pemar@uco.es> \* @version 1.0 \* \* Clase que contiene los metodos para el tratamiento de los paquetes \* de tipo RTP. \*/public class PaqueteRTP { /\*\* \* Tamaño de la cabecera RTP. \*/ static int LONG\_CABECERA = 12;  /\*\* \* Campos de la cabecera RTP. \*/ public int Version; public int Padding; public int Extension; public int CC; public int Marker; public int PayloadType; public int SequenceNumber; public int TimeStamp; public int Ssrc;  /\*\* \* Array de bits de la cabecera RTP. \*/ public byte[] cabecera;  /\*\* \* Tamaño de la carga RTP. \*/ public int longCarga;  /\*\* \* Array de bits de la carga RTP. \*/ public byte[] carga;  /\*\* \* Constructor de un paquete RTP, utilizando los campos de la cabecera y la \* carga. \* \* @param PType \* Tipo de Payload. \* @param Framenb \* Numero de frame. \* @param Time \* Tiempo. \* @param datos \* Datos del video. \* @param longDatos \* Longitud de los datos. \*/ public PaqueteRTP(int PType, int Framenb, int Time, byte[] datos, int longDatos) { /\*\* \* Rellenar los campos de la cabecera por defecto. \*/ Version = 2; Padding = 0; Extension = 0; CC = 0; Marker = 0; Ssrc = 0; /\*\* \* Rellenar los campos que cambian de la cabecera. \*/ SequenceNumber = Framenb; TimeStamp = Time; PayloadType = PType;  /\*\* \* Construir el array de bytes. \*/ cabecera = new byte[LONG\_CABECERA];  /\*\* \* Rellenar con los campos de la cabecera RTP. \*/ cabecera[0] = (byte) (cabecera[0] | Version << 7); cabecera[0] = (byte) (cabecera[0] | Padding << 5); cabecera[0] = (byte) (cabecera[0] | Extension << 4); cabecera[0] = (byte) (cabecera[0] | CC << 3); cabecera[1] = (byte) (cabecera[1] | Marker << 7); cabecera[1] = (byte) (cabecera[1] | PayloadType << 6); cabecera[2] = (byte) (SequenceNumber >> 8); cabecera[3] = (byte) (SequenceNumber & 0xFF); cabecera[4] = (byte) (TimeStamp >> 24); cabecera[5] = (byte) (TimeStamp >> 16); cabecera[6] = (byte) (TimeStamp >> 8); cabecera[7] = (byte) (TimeStamp & 0xFF); cabecera[8] = (byte) (Ssrc >> 24); cabecera[9] = (byte) (Ssrc >> 16); cabecera[10] = (byte) (Ssrc >> 8); cabecera[11] = (byte) (Ssrc & 0xFF); /\*\* \* Rellenar la carga. \*/ longCarga = longDatos; carga = new byte[longDatos];  /\*\* \* Rellenar el array de carga. \*/ carga = datos; } /\*\* \* Constructor de un paquete RTP a partir de un array de bytes. \* \* @param paquete \* Paquete RTP. \* @param longPaquete \* Longitud del paquete RTP. \*/ public PaqueteRTP(byte[] paquete, int longPaquete) { /\*\* \* Rellenar los campos por defecto. \*/ Version = 2; Padding = 0; Extension = 0; CC = 0; Marker = 0; Ssrc = 0; /\*\* \* Comprobar si la longitud total del paquete es menor que la de la \* cabecera. \*/ if (longPaquete >= LONG\_CABECERA) { /\*\* \* Obtener los bits de la cabecera. \*/ cabecera = new byte[LONG\_CABECERA]; for (int i = 0; i < LONG\_CABECERA; i++) { cabecera[i] = paquete[i]; } /\*\* \* Obtener los bits de la carga. \*/ longCarga = longPaquete - LONG\_CABECERA; carga = new byte[longCarga]; for (int i = LONG\_CABECERA; i < longPaquete; i++) { carga[i - LONG\_CABECERA] = paquete[i]; } /\*\* \* Interpretar los campos que cambian de la cabecera. \*/ PayloadType = cabecera[1] & 127; SequenceNumber = cabecera[3] + 256 \* cabecera[2]; TimeStamp = cabecera[7] + 256 \* cabecera[6] + 65536 \* cabecera[5] + 16777216 \* cabecera[4]; } } /\*\* \* Obtener la carga de datos. \* \* @param datos \* Datos del video. \* @return Devuelve el array de bits de carga y su tamaño. \*/ public int getCarga(byte[] datos) { for (int i = 0; i < longCarga; i++) datos[i] = carga[i]; return (longCarga); } /\*\* \* Obtener la longitud de la carga. \* \* @return Devuelve la longitud de la carga. \*/ public int getLongCarga() { return (longCarga); } /\*\* \* Obtener la longitud total del paquete RTP. \* \* @return Devuelve la longitud total del paquete RTP. \*/ public int getLong() { return (longCarga + LONG\_CABECERA); } /\*\* \* Obtener un paquete RTP bien formado. \* \* @param paquete \* Paquete RTP. \* @return Devuelve el paquete y su longitud. \*/ public int getPaquete(byte[] paquete) { /\*\* \* Construye el paquete = cabecera + carga. \*/ for (int i = 0; i < LONG\_CABECERA; i++) paquete[i] = cabecera[i]; for (int i = 0; i < longCarga; i++) paquete[i + LONG\_CABECERA] = carga[i]; /\*\* \* Devuelve la longitud total del paquete. \*/ return (longCarga + LONG\_CABECERA); } /\*\* \* Obtener el tiempo actual. \* \* @return Devuelve el tiempo. \*/ public int getTimestamp() { return (TimeStamp); } /\*\* \* Obtener el numero de secuencia actual. \* \* @return Devuelve el numero de secuencia. \*/ public int getNumeroSecuencia() { return (SequenceNumber); } /\*\* \* Obtener el tipo de carga del paquete RTP. \* \* @return Devuelve el tipo de carga. \*/ public int getTipoCarga() { return (PayloadType); } /\*\* \* Muestra la cabecera. \*/ public void printheader() { // for (int i = 0; i < (LONG\_CABECERA - 4); i++) { // for (int j = 7; j >= 0; j--) // if (((1 << j) & cabecera[i]) != 0) // System.out.print("1"); // else // System.out.print("0"); // System.out.print(" "); // } // System.out.println(); }} |

## Código fuente para la versión 2

Esta versión contiene una interfaz más vistosa y se ha hecho una labor de documentación del código fuente para una mejor extracción de la documentación con JavaDoc más extensa y que pueda servir de ayuda para otros desarrolladores.

### Servidor.java

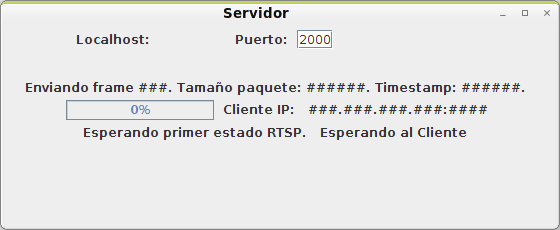


Ilustración . Interfaz Servidor (v2).

|  |
| --- |
| import java.io.\*;  import java.net.\*;  import java.awt.\*;  import java.util.\*;  import java.awt.event.\*;  import javax.swing.\*;  import javax.swing.Timer;  /\*\*  \* @author Raul Perula Martinez <i62pemar@uco.es>  \* @version 1.0  \*  \* Clase del servidor que implementa la ventana grafica junto con las  \* acciones del envio del video a traves de streaming a un cliente.  \*/  public class Servidor extends JFrame implements ActionListener {  /\*\*  \* Variables RTP.  \*/  DatagramSocket socketRTP; // Socket para enviar y recibir datagramas RTP  DatagramPacket pqtEnviar; // Paquete UDP con los frames de video  InetAddress direccionIPCliente; // Direccion IP del Cliente  int puertoDestinoRTP = 0; // Puerto destino para los paquetes  // RTP (proporcionado por el cliente RTSP)  /\*\*  \* GUI.  \*/  JPanel mainPanel;  JPanel configPanel;  JPanel statusPanel;  JLabel label;  JLabel lhost;  /\*\*  \* Etiqueta para el puerto.  \*/  JLabel portLabel;  /\*\*  \* Añadir campo de texto para el puerto del servidor.  \*/  JTextField portnb;  /\*\*  \* Etiqueta para la informacion de estado.  \*/  JLabel statusinfo;  /\*\*  \* Etiqueta para las peticiones.  \*/  JLabel rq;  JLabel clientlabel;  JLabel clientip;  JLabel length;  JProgressBar progress;  /\*\*  \* Variable de video.  \*/  int numImagen = 0; // Numero de imagen transmitida en este momento  StreamVideo video; // VideoStream utilizado para acceder a los frames de  // video  static int TIPO\_MJPEG = 26; // Tipo de carga RTP para video MJPEG  static int PERIODO\_FRAME = 100; // Periodo de frames a transmitir en ms  static int LONG\_VIDEO = 500; // Longitud del video en frames  Timer temporizador; // Temporizador utilizado para enviar las imagenes del  // video al ratio adecuado  byte[] buffer; // Buffer utilizado para almacenar las imagenes a enviar al  // cliente  /\*\*  \* Variables RTSP. Estados RTSP.  \*/  final static int INIT = 0;  final static int READY = 1;  final static int PLAYING = 2;  /\*\*  \* Tipos de mensajes RTSP.  \*/  final static int SETUP = 3;  final static int PLAY = 4;  final static int PAUSE = 5;  final static int TEARDOWN = 6;  static int estado; // Estado del servidor RTSP == INIT o READY o PLAY  Socket RTSPsocket; // Socket utilizado para recibir y enviar mensajes RTSP  /\*\*  \* Canales de entrada y salida.  \*/  static BufferedReader RTSPBufferedReader;  static BufferedWriter RTSPBufferedWriter;  static String ficheroVideo; // Fichero de video solicitado por el cliente  static int RTSP\_ID = 123456; // ID de la sesion RTSP  int RTSPNumSeq = 0; // Numero de secuencia de los mensajes RTSP sin sesion  final static String CRLF = "\n";  /\*\*  \* Constructor.  \*/  public Servidor() {  /\*\*  \* Inicialiar Frame.  \*/  super("Servidor");  /\*\*  \* Inicializar temporizador.  \*/  temporizador = new Timer(PERIODO\_FRAME, this);  temporizador.setInitialDelay(0);  temporizador.setCoalesce(true);  /\*\*  \* Reservar memoria para el buffer de envio.  \*/  buffer = new byte[15000];  /\*\*  \* Manejador para cerrar la ventana principal.  \*/  addWindowListener(new WindowAdapter() {  public void windowClosing(WindowEvent e) {  /\*\*  \* Parar el temporizador y salir.  \*/  temporizador.stop();  System.exit(0);  }  });  /\*\*  \* GUI.  \*/  mainPanel = new JPanel();  statusPanel = new JPanel();  configPanel = new JPanel();  label = new JLabel(  "Enviando frame ###. Tamaño paquete: ######. Timestamp: ######.",  JLabel.CENTER);  lhost = new JLabel("Localhost: ");  portLabel = new JLabel(" Puerto: ");  portnb = new JTextField("2000");  clientlabel = new JLabel(" Cliente IP: ");  clientip = new JLabel(" ###.###.###.###:#### ");  statusinfo = new JLabel(" Esperando primer estado RTSP. ");  rq = new JLabel(" Esperando al Cliente ", JLabel.RIGHT);  length = new JLabel(" ");  progress = new JProgressBar(0, 50000);  progress.setStringPainted(true);  configPanel.add(lhost);  configPanel.add(portLabel);  configPanel.add(portnb);  statusPanel.add(label);  statusPanel.add(length);  statusPanel.add(progress);  statusPanel.add(clientlabel);  statusPanel.add(clientip);  statusPanel.add(statusinfo);  statusPanel.add(rq);  /\*\*  \* Capa de salida.  \*/  mainPanel.setPreferredSize(new Dimension(550, 200));  mainPanel.setLayout(null);  mainPanel.add(configPanel);  mainPanel.add(statusPanel);  configPanel.setBounds(0, 0, 400, 50);  statusPanel.setBounds(20, 50, 500, 400);  getContentPane().add(mainPanel, BorderLayout.CENTER);  }  /\*\*  \* Funcion principal.  \*  \* @param argv  \* Entrada por linea de comandos.  \* @throws Exception  \*/  public static void main(String argv[]) throws Exception {  /\*\*  \* Crear objeto Servidor.  \*/  Servidor servidor = new Servidor();  /\*\*  \* Mostrar GUI.  \*/  servidor.pack();  servidor.setVisible(true);  /\*\*  \* Obtener el puerto RTSP.  \*/  int RTSPport = Integer.parseInt(servidor.portnb.getText());  /\*\*  \* Iniciar la conexion TCP con el cliente para la sesion RTSP.  \*/  ServerSocket socketServidor = new ServerSocket(RTSPport);  servidor.RTSPsocket = socketServidor.accept();  socketServidor.close();  /\*\*  \* Obtener direccion IP del cliente.  \*/  servidor.lhost.setText("Localhost: "  + servidor.RTSPsocket.getLocalSocketAddress() + "");  servidor.direccionIPCliente = servidor.RTSPsocket.getInetAddress();  /\*\*  \* Imprimir la direccion IP del Cliente.  \*/  servidor.clientip.setText(servidor.direccionIPCliente.toString() + ":"  + RTSPport + " (" + servidor.direccionIPCliente.getHostName()  + ") ");  /\*\*  \* Inicializar el estado RTSP.  \*/  estado = INIT;  /\*\*  \* Establecer los canales de entrada y salida.  \*/  RTSPBufferedReader = new BufferedReader(new InputStreamReader(  servidor.RTSPsocket.getInputStream()));  RTSPBufferedWriter = new BufferedWriter(new OutputStreamWriter(  servidor.RTSPsocket.getOutputStream()));  /\*\*  \* Esperar por el mensaje SETUP del cliente.  \*/  int tipoSolicitud;  boolean acabo = false;  while (!acabo) {  tipoSolicitud = servidor.parseSolicitudRTSP(); // Bloqueante  if (tipoSolicitud == SETUP) {  acabo = true;  /\*\*  \* Actualizar estado RTSP.  \*/  estado = READY;  servidor.statusinfo.setText("Nuevo estado RTSP: READY");  /\*\*  \* Enviar respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Inicializar el objeto VideoStream.  \*/  servidor.video = new StreamVideo(ficheroVideo);  /\*\*  \* Inicializar el socket RTP.  \*/  servidor.socketRTP = new DatagramSocket();  }  }  /\*\*  \* Bucle para gestionar las solicitudes RTSP.  \*/  while (true) {  /\*\*  \* Analizar la solicitud.  \*/  tipoSolicitud = servidor.parseSolicitudRTSP(); // Bloqueante  if ((tipoSolicitud == PLAY) && (estado == READY)) {  /\*\*  \* Enviar la respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Iniciar temporizador.  \*/  servidor.temporizador.start();  /\*\*  \* Actualizar estado.  \*/  estado = PLAYING;  servidor.statusinfo.setText("Nuevo estado RTSP: PLAYING");  } else if ((tipoSolicitud == PAUSE) && (estado == PLAYING)) {  /\*\*  \* Enviar la respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Parar temporizador.  \*/  servidor.temporizador.stop();  /\*\*  \* Actualizar estado.  \*/  estado = READY;  servidor.statusinfo.setText("Nuevo estado RTSP: READY");  } else if (tipoSolicitud == TEARDOWN) {  /\*\*  \* Enviar la respuesta.  \*/  servidor.enviarRespuestaRTSP();  /\*\*  \* Parar temporizador.  \*/  servidor.temporizador.stop();  /\*\*  \* Cerrar sockets.  \*/  servidor.RTSPsocket.close();  servidor.socketRTP.close();  System.exit(0);  }  }  }  /\*  \* (non-Javadoc) Manejador del temporizador  \*  \* @see  \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)  \*/  public void actionPerformed(ActionEvent e) {  /\*\*  \* Si el numero de imagen es menor que la longitud del video.  \*/  if (numImagen < LONG\_VIDEO) {  /\*\*  \* Incrementar numImagen.  \*/  numImagen++;  try {  /\*\*  \* Obtener el siguiente frame y su tamaño.  \*/  int longImagen = video.getnextframe(buffer);  /\*\*  \* Construir un paquete RTP con el frame.  \*/  PaqueteRTP paqueteRTP = new PaqueteRTP(TIPO\_MJPEG, numImagen,  numImagen \* PERIODO\_FRAME, buffer, longImagen);  /\*\*  \* Obtener la longitud total del paquete RTP a enviar.  \*/  int longPaquete = paqueteRTP.getLong();  length.setText(" Tamaño paquete: " + longPaquete  + ". Timestamp: " + paqueteRTP.getTimestamp() + ".");  // Print progress  progress.setValue(paqueteRTP.getTimestamp());  /\*\*  \* Recuperar el array de bits del paquete y almacenarlo en un  \* array.  \*/  byte[] bitsPaquete = new byte[longPaquete];  paqueteRTP.getPaquete(bitsPaquete);  /\*\*  \* Enviar el paquete como un DatagramPacket sobre un socket UDP.  \*/  pqtEnviar = new DatagramPacket(bitsPaquete, longPaquete,  direccionIPCliente, puertoDestinoRTP);  socketRTP.send(pqtEnviar);  // System.out.println("Enviando frame #" + numImagen);  /\*\*  \* Mostrar la cabecera.  \*/  paqueteRTP.printheader();  /\*\*  \* Actualizar GUI.  \*/  label.setText("Enviando frame: " + numImagen + ".");  } catch (Exception ex) {  System.out.println("Excepcion 1: " + ex);  System.exit(0);  }  } else {  /\*\*  \* Si se ha alcanzado el final del video, parar temporizador  \*/  temporizador.stop();  }  }  /\*\*  \* Analizar solicitud RTSP.  \*  \* @return Devuelve  \*/  private int parseSolicitudRTSP() {  int tipoSolicitud = -1;  try {  /\*\*  \* Parsear la linea de solicitud y extraer el tipo de solicitud.  \*/  String lineaSolicitud = RTSPBufferedReader.readLine();  rq.setText("Recibido del Cliente: " + lineaSolicitud);  StringTokenizer tokens = new StringTokenizer(lineaSolicitud);  String stringTipoSolicitud = tokens.nextToken();  /\*\*  \* Convertir a una estructura tipoSolicitud.  \*/  if ((new String(stringTipoSolicitud)).compareTo("SETUP") == 0)  tipoSolicitud = SETUP;  else if ((new String(stringTipoSolicitud)).compareTo("PLAY") == 0)  tipoSolicitud = PLAY;  else if ((new String(stringTipoSolicitud)).compareTo("PAUSE") == 0)  tipoSolicitud = PAUSE;  else if ((new String(stringTipoSolicitud)).compareTo("TEARDOWN") == 0)  tipoSolicitud = TEARDOWN;  if (tipoSolicitud == SETUP) {  /\*\*  \* Extraer el fichero de video de la linea de solicitud.  \*/  ficheroVideo = tokens.nextToken();  }  /\*\*  \* Parsear la linea de numero de secuencia.  \*/  String lineaNumSeq = RTSPBufferedReader.readLine();  System.out.println(lineaNumSeq);  tokens = new StringTokenizer(lineaNumSeq);  tokens.nextToken();  RTSPNumSeq = Integer.parseInt(tokens.nextToken());  /\*\*  \* Obtener la ultima linea.  \*/  String ultimaLinea = RTSPBufferedReader.readLine();  System.out.println(ultimaLinea);  if (tipoSolicitud == SETUP) {  /\*\*  \* Extraer el puerto destino RTP de la ultima linea.  \*/  tokens = new StringTokenizer(ultimaLinea);  for (int i = 0; i < 3; i++)  tokens.nextToken(); // no utilizado  puertoDestinoRTP = Integer.parseInt(tokens.nextToken());  }  /\*\*  \* Sino la ultima linea es la linea de identificador de sesion, no  \* se analiza por ahora.  \*/  } catch (Exception ex) {  System.out.println("Excepcion 2: " + ex);  System.exit(0);  }  return (tipoSolicitud);  }  /\*\*  \* Enviar respuesta RTSP.  \*/  private void enviarRespuestaRTSP() {  try {  RTSPBufferedWriter.write("RTSP/1.0 200 OK" + CRLF);  RTSPBufferedWriter.write("CSeq: " + RTSPNumSeq + CRLF);  RTSPBufferedWriter.write("Session: " + RTSP\_ID + CRLF);  RTSPBufferedWriter.flush();  System.out  .println("Servidor RTSP - Enviando respuesta al cliente.");  } catch (Exception ex) {  System.out.println("Excepcion 3: " + ex);  System.exit(0);  }  }  } |

### Cliente.java

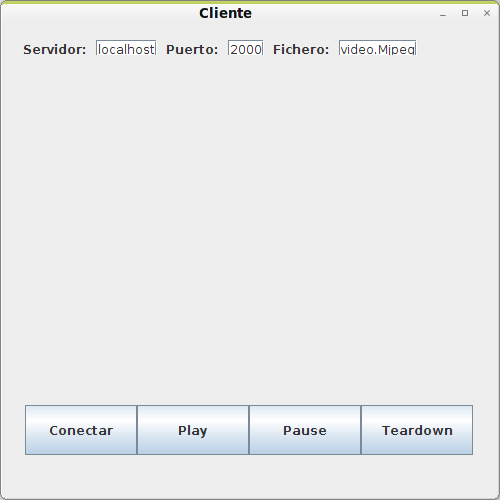


Ilustración . Interfaz Cliente (v2).

|  |
| --- |
| import java.io.\*;import java.net.\*;import java.util.\*;import java.awt.\*;import java.awt.event.\*;import javax.swing.\*;import javax.swing.Timer;/\*\* \* @author Raul Perula Martinez <i62pemar@uco.es> \* @version 1.0 \* \* Clase del cliente que implementa los metodos necesarios para \* conectarse con el servidor, reproducir, pausar el video enviado en \* streaming y finalizar desconectandose de la conexion. \*/public class Cliente { /\*\* \* GUI. \*/  JFrame f = new JFrame("Cliente");  JButton setupButton = new JButton("Conectar"); JButton playButton = new JButton("Play"); JButton pauseButton = new JButton("Pause"); JButton tearButton = new JButton("Teardown"); JPanel mainPanel = new JPanel(); JPanel infoPanel = new JPanel(); JPanel buttonPanel = new JPanel(); JLabel iconLabel = new JLabel(); /\*\* \* Etiqueta para el host. \*/ JLabel hostLabel = new JLabel("Servidor: "); /\*\* \* Añadir campo de texto para el hostname del servidor. \*/ JTextField hostname = new JTextField("localhost"); /\*\* \* Etiqueta para el puerto. \*/ JLabel portLabel = new JLabel(" Puerto: "); /\*\* \* Añadir campo de texto para el puerto del servidor. \*/ JTextField portnb = new JTextField("2000"); /\*\* \* Etiqueta para el nombre del archivo. \*/ JLabel vLabel = new JLabel(" Fichero: "); /\*\* \* Añadir campo de texto para el nombre del archivo. \*/ JTextField vfile = new JTextField("video.Mjpeg"); ImageIcon icon; JLabel countpkt = new JLabel(" "); /\*\* \* RTP variables. \*/ DatagramPacket rcvdp; // Paquete UDP recibido desde el servidor DatagramSocket RTPsocket; // Socket usado para enviar y recibir paquetes UDP static int RTP\_RCV\_PORT = 25000; // Puerto por donde el cliente recibira los // paquetes RTP Timer timer; // Temporizador usado para recibir datos desde el socket UDP byte[] buf; // Bufer usado para almacenar datos recibidos desde el servidor static int countpk = 0; // Contador de paquetes /\*\* \* Variables RTSP. Estados RTSP. \*/ final static int INIT = 0; final static int READY = 1; final static int PLAYING = 2; static int state; // Estado RTSP == INIT o READY o PLAYING Socket RTSPsocket; // Socket usado para enviar/recibir mensajes RTSP /\*\* \* Flujos de entrada y salida. \*/ static BufferedReader RTSPBufferedReader; static BufferedWriter RTSPBufferedWriter; static String VideoFileName; // Fichreo de video para pedir al servidor int RTSPSeqNb = 0; // Numero de secuencia de los mensajes RTSP sin la sesion int RTSPid = 0; // ID de la sesion RTSP (dada por el servidor RTSP) final static String CRLF = "\n"; /\*\* \* Variables para la interfaz. \*/ static int setupstage = 0; static int sizem; static int tc = 1; static int dt = 1; /\*\* \* Constantes de video. \*/ static int MJPEG\_TYPE = 26; // Tipo de RTP para video MJPEG /\*\* \* Constructor. \*/ public Cliente() { /\*\* \* Construccion de la GUI. \*/ /\*\* \* Frame. \*/ f.addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent e) { System.exit(0); } }); /\*\* \* Informacion y configuracion. \*/ /\*\* \* Añadir las etiquetas y los campos de texto. \*/ infoPanel.add(hostLabel); infoPanel.add(hostname); infoPanel.add(portLabel); infoPanel.add(portnb); infoPanel.add(vLabel); infoPanel.add(vfile); infoPanel.add(countpkt); /\*\* \* Botones. \*/ buttonPanel.setLayout(new GridLayout(1, 0)); buttonPanel.add(setupButton); buttonPanel.add(playButton); buttonPanel.add(pauseButton); buttonPanel.add(tearButton); setupButton.addActionListener(new setupButtonListener()); playButton.addActionListener(new playButtonListener()); pauseButton.addActionListener(new pauseButtonListener()); tearButton.addActionListener(new tearButtonListener()); /\*\* \* Etiqueta de imagen mostrada. \*/ iconLabel.setIcon(null); /\*\* \* Capa de salida del frame. \*/ mainPanel.setLayout(null); mainPanel.add(infoPanel); mainPanel.add(iconLabel); mainPanel.add(buttonPanel); infoPanel.setBounds(-20, 10, 540, 20); iconLabel.setBounds(50, 10, 500, 400); buttonPanel.setBounds(20, 380, 450, 50); f.getContentPane().add(mainPanel, BorderLayout.CENTER); f.setSize(new Dimension(500, 500)); f.setVisible(true); /\*\* \* Inicializacion del temporizados. \*/ timer = new Timer(20, new timerListener()); timer.setInitialDelay(0); timer.setCoalesce(true); /\*\* \* Almacenamiento de suficiente memoria para el bufer usado para recibir \* datos desde el servidor. \*/ buf = new byte[15000]; } /\*\* \* Funcion principal. \* \* @param argv \* Lista de argumentos en linea de comandos. \* @throws Exception \*/ public static void main(String argv[]) throws Exception { /\*\* \* Creacion de un objeto Cliente. \*/ Cliente cliente = new Cliente(); /\*\* \* Obtener puerto RTSP del servidor y la direccion IP desde linea de \* comandos. \*/ int RTSP\_server\_port = Integer.parseInt(cliente.portnb.getText()); String ServerHost = cliente.hostname.getText(); InetAddress ServerIPAddr = InetAddress.getByName(ServerHost); /\*\* \* Obtener nombre de fichero de video. \*/ VideoFileName = cliente.vfile.getText(); /\*\* \* Establecer una conexion TCP con el servidor para intercambiar \* mensajes RTSP. \*/ cliente.RTSPsocket = new Socket(ServerIPAddr, RTSP\_server\_port); /\*\* \* Poner los flujos de entrada y salida. \*/ RTSPBufferedReader = new BufferedReader(new InputStreamReader( cliente.RTSPsocket.getInputStream())); RTSPBufferedWriter = new BufferedWriter(new OutputStreamWriter( cliente.RTSPsocket.getOutputStream())); /\*\* \* Inicializar estado RTSP. \*/ state = INIT; } /\*\* \* Funciones de los botones. \*/ /\*\* \* Funcion para el boton Setup. \*/ class setupButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { setupstage++; countpk = 0; if (state == INIT) { /\*\* \* Inicializar un RTPsocket no bloqueante que sera usado para \* recibir los datos. \*/ try { /\*\* \* Construir un nuevo DatagramSocket para recibir los \* paquetes RTP desde el servidor, en el puerto \* RTP\_RCV\_PORT. \*/ RTPsocket = new DatagramSocket(RTP\_RCV\_PORT); /\*\* \* Poner el valor de TimeOut a 5ms. \*/ RTPsocket.setSoTimeout(5); } catch (SocketException se) { System.out.println("Excepcion de socket: " + se); System.exit(0); } /\*\* \* Inicializar el numero de secuencia RTSP. \*/ RTSPSeqNb = 1; /\*\* \* Enviar mensaje SETUP al servidor. \*/ send\_RTSP\_request("SETUP"); /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor nvalida"); else { /\*\* \* Cambiar el estado RTSP e imprimir el nuevo stado. \*/ state = READY; System.out.println("Nuevo estado RTSP: READY"); } } } } /\*\* \* Funcion para el boton Play. \*/ class playButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* ava.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { if (state == READY) { /\*\* \* Incremento del numero de secuencia RTSP. \*/ RTSPSeqNb++; /\*\* \* Enviar mensaje PLAY al servidor. \*/ send\_RTSP\_request("PLAY"); /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor Invalida"); else { /\*\* \* Cambiar el estado RTSP e imprimir el nuevo estado. \*/ state = PLAYING; System.out.println("Nuevo estado RTSP: PLAYING"); /\*\* \* Activar el temporizador. \*/ timer.start(); } } } } /\*\* \* Funcion del boton Pause. \*/ class pauseButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { if (state == PLAYING) { /\*\* \* Incrementar el numero de secuencia RTSP. \*/ RTSPSeqNb++; /\*\* \* Enviar mensaje PAUSE al servidor. \*/ send\_RTSP\_request("PAUSE"); /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor Invalida"); else { /\*\* \* Cambiar estado RTSP e imprimir el nuevo estado. \*/ state = READY; System.out.println("Nuevo estado RTSP: READY"); /\*\* \* Parar el temporizador. \*/ timer.stop(); } } } } /\*\* \* Funcion del boton Teardown. \*/ class tearButtonListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { /\*\* \* Incrementar el numero de secuencia RTSP. \*/ RTSPSeqNb++; /\*\* \* Enviar mensaje TEARDOWN al servidor. \*/ send\_RTSP\_request("TEARDOWN"); /\*\* \* Esperar la respuesta. \*/ if (parse\_server\_response() != 200) System.out.println("Respuesta del Servidor Invalida"); else { /\*\* \* Cambiar estado RTSP e imprimir el nuevo estado. \*/ state = INIT; System.out.println("Nuevo estado RTSP: INIT"); /\*\* \* Parar el temporizador. \*/ timer.stop(); /\*\* \* Salir. \*/ System.exit(0); } } } /\*\* \* Funcion del temporizador. \*/ class timerListener implements ActionListener { /\* \* (non-Javadoc) \* \* @see \* java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent \* ) \*/ public void actionPerformed(ActionEvent e) { /\*\* \* Construir un DatagramPacket para recibir datos desde el socket \* UDP. \*/ rcvdp = new DatagramPacket(buf, buf.length); try { /\*\* \* Recibir el DP desde el socket. \*/ RTPsocket.receive(rcvdp); /\*\* \* Crear un objeto PaqueteRTP desde el DP. \*/ PaqueteRTP rtp\_packet = new PaqueteRTP(rcvdp.getData(), rcvdp .getLength()); /\*\* \* Imprimir el total de paquetes por segundo. \*/ sizem = sizem + rtp\_packet.getLongCarga(); dt = tc - rtp\_packet.getTimestamp(); tc = rtp\_packet.getTimestamp(); countpkt.setText(" " + (sizem / tc) + "KB/s"); /\*\* \* Imprimir los campos de las cabeceras importantes del paquete \* RTP recibido. \*/ System.out .println("Recibido paquete RTP con Numero Secuencia # " + rtp\_packet.getNumeroSecuencia() + " TimeStamp " + rtp\_packet.getTimestamp() + " ms, de tipo " + rtp\_packet.getTipoCarga()); /\*\* \* Imprimir la cabecera bitstream. \*/ rtp\_packet.printheader(); /\*\* \* Obtener el bitstream desde el objeto PaqueteRTP. \*/ int payload\_length = rtp\_packet.getLongCarga(); byte[] payload = new byte[payload\_length]; rtp\_packet.getCarga(payload); /\*\* \* Obtener una objeto imagen desde el bitstream. \*/ Toolkit toolkit = Toolkit.getDefaultToolkit(); Image image = toolkit.createImage(payload, 0, payload\_length); /\*\* \* Mostrar la imagen como un objeto ImageIcon. \*/ icon = new ImageIcon(image); iconLabel.setIcon(icon); } catch (InterruptedIOException iioe) { // System.out.println("Nada que leer"); } catch (IOException ioe) { System.out.println("Excepcion capturada: " + ioe); } } } /\*\* \* Parsear respuesta del servidor. \* \* @return Devuelve el codigo de respuesta. \*/ private int parse\_server\_response() { int reply\_code = 0; try { /\*\* \* Parsear estado de la linea y extraer el reply\_code. \*/ String StatusLine = RTSPBufferedReader.readLine(); System.out.println("Cliente RTSP - Recibido desde el servidor:"); System.out.println(StatusLine); StringTokenizer tokens = new StringTokenizer(StatusLine); tokens.nextToken(); // Saltar al siguiente RTSP reply\_code = Integer.parseInt(tokens.nextToken()); /\*\* \* Si el reply\_code esta OK se obtiene e imprime las 2 siguiente \* lineas. \*/ if (reply\_code == 200) { String SeqNumLine = RTSPBufferedReader.readLine(); System.out.println(SeqNumLine); String SessionLine = RTSPBufferedReader.readLine(); System.out.println(SessionLine); /\*\* \* Si el estado == INIT obtiene el Id de Sesion desde la \* SessionLine. \*/ tokens = new StringTokenizer(SessionLine); tokens.nextToken(); // Saltar a la siguiente sesion RTSPid = Integer.parseInt(tokens.nextToken()); } } catch (Exception ex) { System.out.println("Excepcion capturada: " + ex); System.exit(0); } return (reply\_code); } /\*\* \* Enviar peticion RTSP. \* \* @param request\_type \* Tipo de peticion. \*/ private void send\_RTSP\_request(String request\_type) { try { /\*\* \* Usar el RTSPBufferedWriter para escribir el socket RTSP. Escribir \* la linea de peticion. \*/ RTSPBufferedWriter.write(request\_type + " " + VideoFileName + " RTSP/1.0" + CRLF); /\*\* \* Escribir la linea CSeq. \*/ RTSPBufferedWriter.write("CSeq: " + RTSPSeqNb + CRLF); /\*\* \* Comprobar si el request\_type es igual a "SETUP" y en ese caso \* escribir el Transport: linea de aviso al servidor del puerto \* usado para recibir los paquetes RTP, RTP\_RCV\_PORT. \*/ if ((new String(request\_type)).compareTo("SETUP") == 0) RTSPBufferedWriter.write("Transport: RTP/UDP; client\_port= " + RTP\_RCV\_PORT + CRLF); else RTSPBufferedWriter.write("Session: " + RTSPid + "\n"); RTSPBufferedWriter.flush(); } catch (Exception ex) { System.out.println("Excepcion capturada: " + ex); System.exit(0); } }} |

### VideoStream.java

Este código es igual al de la versión anterior.

### PaqueteRTP.java

Este código es igual al de la versión anterior.

# Futuras mejoras

Aunque la aplicación ya de por sí está bastante completa en lo que a la funcionalidad que se buscaba, no obstante, es todavía muy mejorable y la parte más importante para la transmisión de videoconferencias no ha sido implementada.

Con lo que, podría haber mejoras en:

* Implementar la captación de imágenes desde la webcam para su seguida transmisión.
* Agregar funcionalidad a la interfaz gráfica, por ejemplo, añadiendo un aspecto más profesional y asemejándolo a un reproductor de escritorio al estilo del programa VLC (simple pero con mucha funcionalidad).
* Implementar y adaptar el proyecto y la aplicación a un entorno web o para un dispositivo móvil al estilo del famoso programa Skype.

Todas estas posibles mejoras harían que la aplicación tomase un ente más profesional y que pueda verse como una aplicación de mercado.

# Bibliografía

1. “Apuntes Moodle”. UCO. Última visita: Mayo-2011. Enlace: <http://www3.uco.es/moodle/>
2. “Conexiones Cliente-Servidor mediante sockets en Java”. Casifiablo.net. Última visita: Mayo-2011. Enlace: <http://casidiablo.net/java-socket-chat-basico/>
3. “CLIENTE/SERVIDOR TCP/IP”. Tutorial de Java. Última visita: Mayo-2011. Enlace: <http://sunsite.dcc.uchile.cl/java/docs/JavaTut/Cap9/cli_ser.html>
4. “Servidor Streaming Dolphin”. Google Code. Última visita: Mayo-2011. Enlace: <http://code.google.com/p/servidor-streaming-rtp-rstp-java/>
5. “Servidor Streaming Java”. Debugmodeon. Última visita: Mayo-2011. Enlace: <http://es.debugmodeon.com/debate/servidor-streaming-java>
6. “java.net.RTP”. Report de Colombia. Última visita: Mayo-2011. Enlace: <http://www.cs.columbia.edu/~hgs/teaching/ais/1998/projects/java_rtp/report.html>
7. “Java SE Desktop Technologies. Descarga de JMF”. Oracle. Última visita: Mayo-2011. Enlace: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html>
8. “JMF, Usando la webcam desde JAVA y guardando una Imagen”. Wordpress. Última visita: Mayo-2011. Enlace: <http://cmop17.wordpress.com/2010/01/14/jmf-usando-la-webcam-desde-java-y-guardando-una-imagen/>
9. “Real Time Streaming Protocol”. Wikipedia. Última visita: Mayo-2011. Enlace: <http://es.wikipedia.org/wiki/Real_Time_Streaming_Protocol>
10. “RFC: Real Time Streaming Protocol (RTSP)”. IETF. Última visita: Mayo-2011. Enlace: <http://www.ietf.org/rfc/rfc2326.txt>
11. “Real-time Transport Protocol”. Wikipedia. Última visita: Mayo-2011. Enlace: <http://es.wikipedia.org/wiki/Real-time_Transport_Protocol>
12. “Java Media Framework”. UMA. Última visita: Mayo-2011. Enlace: <http://www.lcc.uma.es/~pinilla/TutorialJMF/Procesamiento.htm>

# Apéndice A. Instalación y desinstalación

Para la instalación del programa se podrá utilizar el IDE Eclipse. Importando el proyecto ya tendrá instalada la aplicación. Si no tiene el IDE o no desea tenerlo, siempre existe la posibilidad de hacerlo de forma manual. En ese caso no tendrá nada más que compilar los archivos fuente y así obtendrá los ficheros ejecutables disponibles para su ejecución y prueba.

Si desea desinstalarla, no tendrá nada más que eliminar el proyecto de Eclipse permanentemente o eliminar la carpeta contenedora del proyecto.

# Apéndice B. Manual de usuario

Para la ejecución correcta del programa tendrá que seguir los siguientes pasos:

1. Ejecute el programa Servidor para la posible recepción de clientes que quieran comunicarse.
2. Para ***ejecutar el Servidor*** tendrá que:
   1. En la versión 1:
      1. Ejecutar el servidor especificando por línea de comandos el puerto RSTP. Por lo general, el puerto que suele usarse es el 2000.

|  |
| --- |
| **$ *Java Servidor 2000*** |

Servidor_001.png

Ilustración . Servidor. Momento inicial (v1).

Servidor_003.png

Ilustración . Servidor. En reproducción (v1).

* 1. En la versión 2:
     1. Ejecutar el servidor. El puerto ya está especificado. Si se desea cambiar el puerto solo deberá modificar el campo de texto destinado a ello.

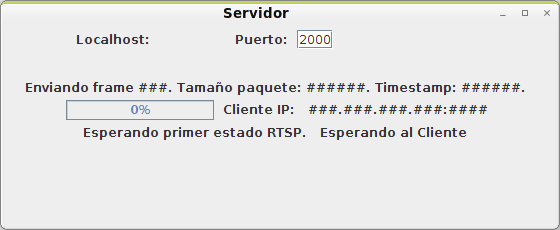
****

Ilustración . Servidor. Momento inicial (v2).

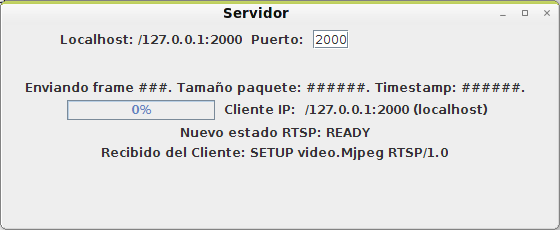


Ilustración . Servidor. Preparado para reproducción (v2).

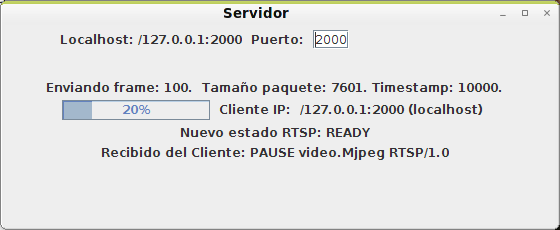


Ilustración . Servidor. En reproducción (v2).

1. Una vez ejecutado el Servidor procedamos a ***ejecutar el Cliente***. Para ello tendrá que:
   1. En la versión 1:
      1. Ejecutar el cliente especificando por línea de comandos el nombre del host, el puerto RSTP y el nombre del fichero de vídeo a usar. Por lo general, el nombre del host será “localhost” para pruebas locales. El puerto será el 2000 que es el que normalmente está destinado a RSTP. Y el nombre del fichero de vídeo dará igual, siempre y cuando se encuentre en formato MJPEG.

|  |
| --- |
| **$ *java Cliente localhost 2000 video.Mjpeg*** |

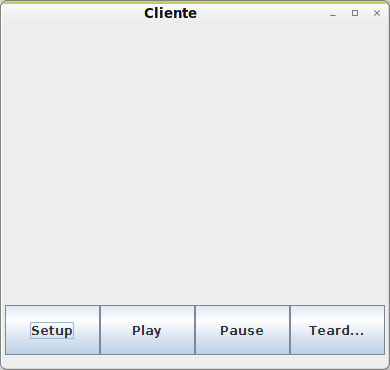


Ilustración . Cliente. Preparado para pulsar Setup (v1).

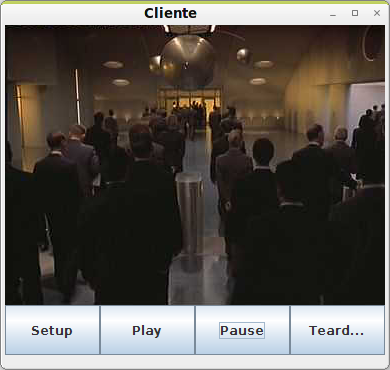


Ilustración . Cliente. En reproducción (v1).

* 1. En la versión 2:
     1. Ejecutar el cliente. Por defecto ya vienen especificados los valores que podrá tomar. Si se desea modificar alguno de estos, solo deberá acceder al campo de texto correspondiente y modificar el valor del mismo.

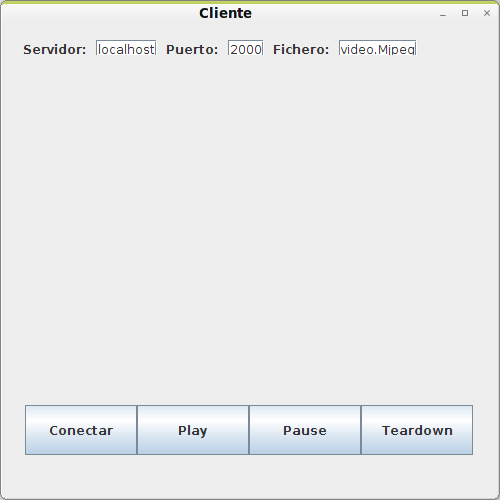


Ilustración . Cliente. Preparado para pulsar sobre Conectar (v2).



Ilustración . Cliente. En reproducción (v2).

1. Una vez ejecutados ambos podrá realizar las siguientes acciones (ambas versiones):
   1. ***Conectar*** el Cliente al Servidor. Una vez realizado esto tendrá la opción de realizar el resto de opciones.
   2. ***Reproducir*** el contenido del vídeo.
   3. ***Pausar*** el vídeo.
   4. ***Parar*** el vídeo y ***salir*** de la aplicación Cliente.
2. Finalizado el Cliente podrá realizar dos opciones:
   1. ***Ejecutar*** de nuevo un Cliente.
   2. ***Finalizar*** el Servidor.