ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Diseño e implementación de *Multiple Description Coding* para aplicaciones P2P de vídeo bajo demanda

AUTOR: José Antonio Díaz Mateo
DIRECTOR: Juan Pedro Muñoz Gea

Julio / 2010

Autor	José Antonio Díaz Mateo
E-mail del Autor	joseantonio.diaz85@gmail.com
Director	Juan Pedro Muñoz Gea
E-mail del Director	juanp.gea@upct.es
Título del PFC	Diseño e implementación de *Multiple Description Coding* para aplicaciones P2P de vídeo bajo demanda
Descriptores	Vídeo bajo demanda; P2P; streaming.

Resumen

El uso de las redes P2P ha sufrido un gran crecimiento en los últimos años. Desde sus primeros usos para intercambiar archivos, su uso se ha extendido hacia otros servicios. Uno de ellos es la transmisión de vídeo bajo demanda. La mayoría de estas aplicaciones tratan a todos los *peers* por igual independientemente del ancho de banda disponible y de su capacidad de procesamiento. El *Multiple Description Coding* (MDC) permite la transmisión de vídeo en *streaming* por una red P2P en la que sus peers son heterogéneos, es decir, cada uno tiene unas características (ancho de banda, CPU) y dependiendo de ellas recibirá el vídeo con más o menos calidad.

En este proyecto se ha llevado a cabo el desarrollo de una aplicación cliente – servidor, capaz de transmitir video en tiempo real a alta calidad donde el receptor es quien decide finalmente si lo reproducirá o no a dicha calidad, limitando el número de flujos de video que utiliza para la reconstrucción del vídeo final en función de tres posibles opciones (Alta, Media o Baja). Además el sistema soporta concurrencia, donde el número máximo de clientes simultáneos que se pueden manejar es configurable.

Titulación	Ingeniería Técnica de Telecomunicación, especialidad Telemática
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Julio - 2010

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1
Conceptos básicos

1.1. Introducción

El uso de estas redes P2P para distribuir video en una red IP (Internet) tiene unas ventajas muy claras, aunque de todas ellas sobresale una muy visiblemente: el ahorro de ancho de banda que utiliza el servidor del video, ya que no tiene que distribuir el video a todos los clientes (peers), sino que lo distribuye a un número concreto de ellos y cada peer que recibe el video lo distribuye a otros. Esta arquitectura, sin embargo, tiene un gran inconveniente: la complejidad del sistema, ya que resulta más sencillo conectarse a un servidor y recibir el video, que buscar peers y recibir un "pedazo" del video de cada uno.

Existen varios programas que funcionan como "televisiones P2P", que conectan con varios canales, de los cuales el más extendido quizás sea el "Coolstreaming". Estos programas han aparecido bastante recientemente, y todos tienen una estructura muy similar: un servidor coge un video, lo trocea en paquetes y lo ofrece a la red, luego un peer recibe este video, lo recompone y lo puede reproducir. Este proceso es sencillo y tiene unas características inherentes: si un paquete no llega, no se puede reproducir ese pedazo del video y, a parte, el peer ve el video a la calidad a la que lo emite el servidor, por lo que se puede decir que se trata a todos los peers por igual. Ahora bien, en la práctica esto no es así, ya que cada vez aparecen más dispositivos capaces de conectarse a Internet, desde un terminal móvil hasta una televisión de alta definición a través de Apple TV (p.ej.). Estos dos dispositivos tienen unas diferencias abismales. Mientras que el terminal móvil no necesita mucha calidad y suele estar asociado a un bajo ancho de banda, la televisión de alta definición necesita una gran calidad del video y puede tener asociado bastante ancho de banda.

Para solventar este problema existen varios mecanismos, como el Layered Coding (LC) o Múltiple Description Coding (MDC). Estos lo que hacen es dividir el video en "capas" y distribuirlas por separado por la red y el peer reproducirá el video con más o menos calidad según el número de capas que reciba. Pero estos dos mecanismos tienen diferencias sustanciales. El LC divide el video en diferentes tipos de capas: una base y una o más de información extra. El peer necesita la capa base para reproducir el video a su mínima calidad y añadiendo capas de información extra gana calidad. Si se pierde un paquete de la capa base, el video no se puede reproducir durante ese tiempo. En cambio, el MDC divide el video en n capas iguales. Sólo hace falta una (da igual cual, puede ser cualquiera) para reproducir el video a su mínima calidad y se gana en calidad recibiendo más capas, hasta la calidad máxima que se consigue recibiendo las n capas. Si se pierde un paquete, perderemos calidad en el video durante ese instante. Es sobre este último mecanismo (MDC) sobre el que se va a basar el PFC.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación cliente – servidor que permita el intercambio de video en tiempo real. Para ello, se llevarán a cabo las siguientes tareas:

Conceptos Básicos

Primeramente se estudiarán los formatos de la familia YUV y los algoritmos MDC para conocer las bases teóricas sobre las que se sustenta el proyecto, ofreciendo una descripción detallada sobre los dos grandes grupos dentro de los YUV (formatos planos y empaquetados), así como las diferentes formas de

llevar a cabo la implementación del algoritmo MDC, definiendo posibles opciones y formas de tratar el vídeo en función de las mismas.

Desarrollo Teórico

En este apartado, nos centraremos en la aplicación cliente – servidor. Se describirán con detalle los aspectos teóricos del desarrollo de la misma a partir de una serie de esquemas que mostrarán un punto de vista genérico de la aplicación a desarrollar, así como cada una de las partes por separado (cliente y servidor), ofreciendo una mejor comprensión de lo que se pretende realizar.

Implementación

En la implementación se describirán las herramientas utilizadas, además de las posibles opciones que se han descartado, tanto en términos de lenguajes de programación como en posibles herramientas externas. Además, se describirá con detalle cada uno de los módulos que componen la aplicación, atendiendo a los criterios teóricos anteriormente definidos.

Evaluación y Prestaciones

Aquí se realizarán una serie de pruebas para extraer conclusiones sobre el rendimiento de la aplicación, mediante una comparativa entre las diferentes funcionalidades que ofrece ésta, a la hora de transmitir y reproducir el video, para conocer cuáles son los mejores resultados en función del algoritmo utilizado y la calidad seleccionada.

Conclusiones

Aquí se exponen las conclusiones generales del proyecto, resumiendo brevemente todo lo que se ha hecho durante el mismo. Además, también se trata un pequeño apartado en el que se proponen posibles líneas futuras por las que puede encaminarse el proyecto.

1.3. YUV

Esta familia de formatos toma como base el hecho de que el ojo humano es más sensible a la luz que al color a la hora de efectuar la compresión del video. Por ello, estos formatos se basan en la utilización de tres componentes que representan la luminosidad de un pixel y el color del mismo. Para realizar esta representación se utilizan una componente para la luminosidad, que llamaremos Y (luminancia), y dos para el color, que denominaremos U y V (crominancia) respectivamente.

Existen dos grandes grupos dentro de los formatos YUV, los formatos a los que se denominan empaquetados y aquellos conocidos como formatos planos.

1.3.1. El submuestreo y su notación

Antes de entrar en los diferentes tipos de formatos YUV, es conveniente conocer el concepto de submuestreo y como se refleja éste a la hora de denominar a los diferentes tipos de formatos YUV. Normalmente nos encontraremos con la notación X:X:X como parte del nombre de un formato YUV, o bien, como parte de la descripción y los detalles de éste.

Una imagen se puede ver como una matriz cuyos elementos son los pixeles de la misma, cada uno de los cuales, contiene las componentes Y, U y V. El submuestreo hace referencia a la frecuencia con la que se toman muestras de dicha matriz.

Existen dos tipos de submuestreo: el Horizontal y el Vertical. El Submuestreo Horizontal hace referencia al número de muestras que se toman por línea, mientras que el Submuestreo Vertical, hace referencia al número de líneas sobre el que se tomarán dichas muestras. Por ello, cuando por ejemplo, nos dicen que un formato es 4:2:2, nos están informando sobre el numero de muestras que se toman de cada una de las componentes Y, U y V, siendo en este caso concreto, la totalidad para las componentes de luminancia, y tan sólo la mitad para las componentes de crominancia.

	Horizontal	Vertical
Y Sample Period	1	1
V Sample Period	2	2
U Sample Period	2	2

Figura 1.1 – Submuestro y notación de los formatos YUV.

Como se puede ver en la Figura 1.1, cuando tenemos un formato 4:2:2, lo que estamos indicando, es que se toman muestras de Crominancia cada dos pixeles y cada dos líneas de la imagen, mientras que en el caso de la luminancia se toman todas las muestras.

1.3.2. Formatos empaquetados o de componentes conjuntas

En los formatos empaquetados todas las componentes se encuentran dentro de un mismo array almacenadas en grupos que se denominan Macropixeles. Estos grupos engloban dentro de sí varios pixeles de la imagen. A continuación se explicarán los diferentes formatos YUV que siguen esta lógica.

1.3.2.1. Formatos 4:4:4

AYUV. Este formato utiliza 8 bits para representar cada una de las muestras de cada componente, es decir, tendremos 8 bits por cada muestra de luminancia (componente Y), y también 8 bits para cada muestra de las componentes asociadas a la crominancia (tanto para la componente U como la V). Además se maneja también un valor de mezcla Alfa (A) por pixel. El orden de las componentes es AYUV, tal como indica el propio nombre del formato.

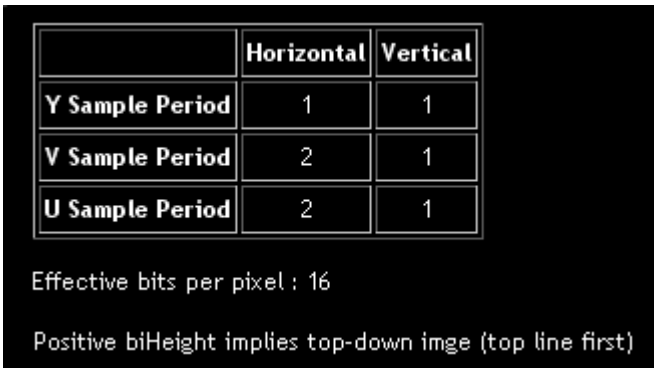
IYU2. Éste es el formato que utilizan las cámaras digitales IEEE 194 en modo 0. Este formato es como el anterior, pero en este caso se usan 24 bits por muestra. El orden de almacenamiento de las componentes es el de la Figura 1.2.

Byte	0	1	2	3	4	5
Sample	$U_{(K+0)}$	$Y_{(K+0)}$	$V_{(K+0)}$	$U_{(K+1)}$	$Y_{(K+1)}$	$V_{(K+1)}$

Figura 1.2 – Orden de bytes para IYU2.

1.3.2.2. Formatos 4:2:2

UYVY. Se trata de un formato 4:2:2 que utiliza 16 bits para representar cada pixel, constituyéndose cada Macropixel de 2 pixeles de la imagen original, que como ya se explicó se trata de la forma en que se agrupan los pixeles de la imagen en los formatos empaquetados. A continuación se puede ver una ilustración que refleja todo lo explicado.



	Horizontal	Vertical
Y Sample Period	1	1
V Sample Period	2	1
U Sample Period	2	1

Effective bits per pixel : 16

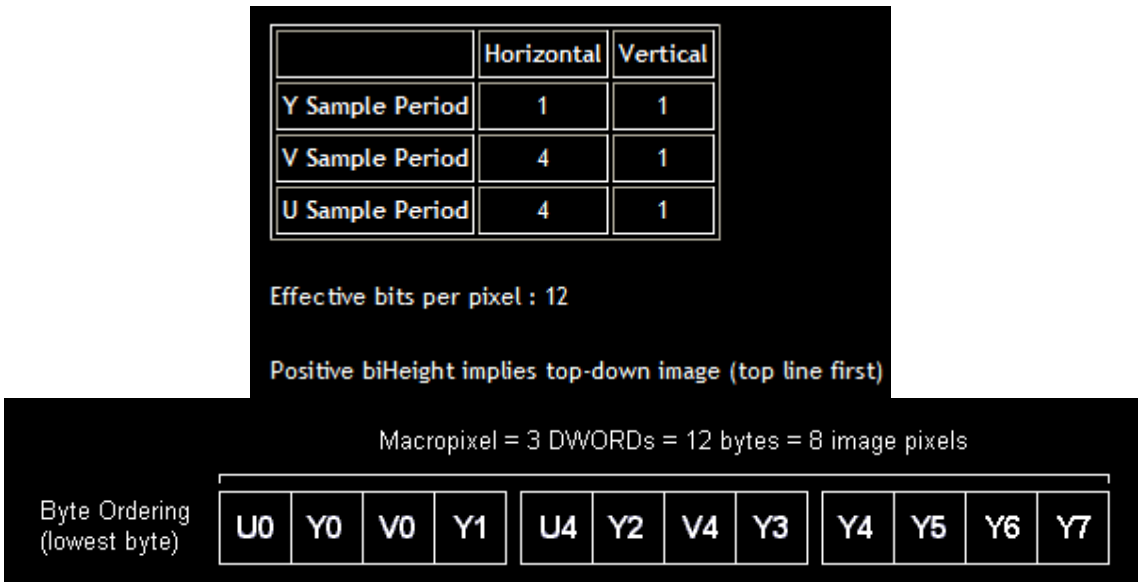
Positive biHeight implies top-down imge (top line first)

Figura 1.3 – Estructura de componentes.

Figura 1.4 – Organización de las distintas componentes dentro de los datos en bruto en formatos 4:2:2.

1.3.2.3. Formatos 4:1:1

Y41P. Se trata de un formato registrado como estándar PCI. Su estructura en cuanto a l orden de las componentes, bits efectivos y periodo para el submuestreo se puede ver en la Figura 1.4.



	Horizontal	Vertical
Y Sample Period	1	1
V Sample Period	4	1
U Sample Period	4	1

Effective bits per pixel : 12

Positive biHeight implies top-down image (top line first)

Figura 1.5 – Orden de bytes, periodo para submuestreo y bits efectivos por pixel para Y41P.

IY41 e Y41T. Ambos formatos toman como base la estructura de Y41P. IY41, únicamente se diferencia de Y41P por el hecho de utilizar entrelazado de componentes (0,2,4,… 1,3,5,… en lugar de 0,1,2,3,…). Por su parte, Y41T, es una versión de Y41P con la diferencia de que el bit menos significativo de cada componente Y, se usa para crear un canal cromático. Este canal cromático nos sirve para dar transparencia a nuestra imagen en función de si está activo o no, es decir, si el bit menos

significativo de una componente Y está a uno, el pixel al que corresponde la componente Y se mostrará de lo contrario, dicho pixel será transparente.

1.3.2.4. Formatos 2:1:1

Y211. Actualmente no existe ningún dispositivo (que se sepa) que tenga como formato de salida uno de este tipo. La principal diferencia con las familias 422 y 411, es que en este caso si se elimina una parte de las muestras de luminancia, tomándolas cada dos pixeles, en lugar de hacerlo en todos ellos Los pixeles de la imagen se agrupan en bloques de cuatro, es decir, cada macropixel contiene cuatro pixeles de la imagen original.

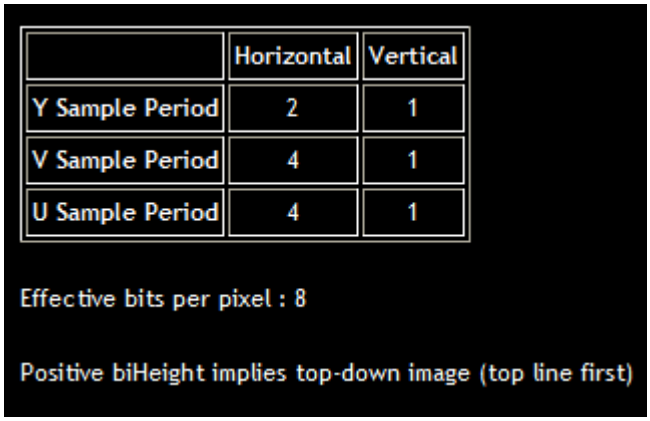


Figura 1.6 – Periodos para el submuestreo y bits efectivos por pixel para Y211.

Figura 1.7 – Orden de bytes para Y211.

1.3.3. Formatos planos o de componentes separadas

Los Formatos Planos o de Componentes Separadas, son aquellos que almacenan las muestras asociadas a la componente de luminancia (Y) y las muestras asociadas a cada una de las componentes de crominancia (U y V) por separado mediante el uso de matrices individuales por componente. De este modo tendremos una matriz en la que sólo tendremos muestras de luminancia (Y), otra en la que solo tendremos muestras asociadas a la componente de crominancia U y por último, otra en la que sólo tendremos muestras asociadas a la componente de crominancia V. Ahora pasaremos a describir las características de los diferentes formatos YUV que se sirven de esta estrategia como estructura de almacenamiento

1.3.3.1. Formatos 4:2:2

YV12. Se utiliza en muchos Códecs MPEG. Su estructura se compone de una matriz para las muestras de luminancia de dimensión N x M y sendas matrices para las muestras asociadas a las componentes de crominancia cuya dimensión es (n/2) x (n/2).

	Horizontal	Vertical
Y Sample Period	1	1
V Sample Period	2	2
U Sample Period	2	2

Figura 1.8 – Periodos para el submuestreo en YV12.

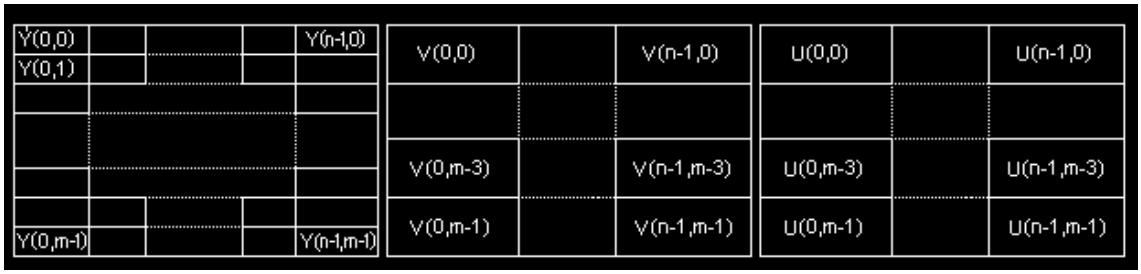


Figura 1.9 – Estructura de YV12.

YV16. Se trata de una versión de YV12 con mayor resolución cromática, cuya composición se constituye de una matriz de dimensión N x M para la luminancia, y otras dos de dimensión (N/2) x M para cada una de las componentes cromáticas. Esto quiere decir que el periodo de submuestreo es menor, tal y como se refleja en la Figura 1.10.

	Horizontal	Vertical
Y Sample Period	1	1
V Sample Period	2	1
U Sample Period	2	1

Figura 1.10 – Periodos para el submuesreo en YV16.

CLPL. Se trata de otro formato de la familia 422 con las típicas dimensiones para sus matrices (Y(N x N), U((N/2) x (N/2)) y V((N/2) x (N/2))), siendo el orden de las matrices YUV en lugar de YVU, como en YV12. Tiene la peculiar diferencia de que utiliza un segundo nivel de direccionamiento indirecto ofreciéndonos un mapa de direccionamiento general que es el que nos da acceso a los punteros que direccionan a los elementos de las matrices, es decir, cada puntero del citado mapa general de direccionamiento, contiene a su vez tres punteros que representan los elementos de las tres matrices Y, U y V.

IMC1 e IMC2. Ambos son versiones de YV12 con las mismas dimensiones de matriz para cada componente, pero con una salvedad, la diferencia principal entre estos dos YUV 422, radica en la forma de organizar las matrices en memoria, que no se queda en un simple intercambio en el orden, es más tal intercambio no existe.

En IMC1, lo que se hace es añadir relleno a las matrices U y V para ajustar el tamaño haciéndolo cuadrar con el de la matriz Y. De esta manera las matrices se pueden considerar parte de una matriz mayor que las contiene. En cambio, en IMC2, no se añade relleno, realizando en su lugar una alineación de las matrices U y V de tal forma que ambas son adyacentes a la matriz Y en memoria. A continuación se pueden ver unas ilustraciones que aclaran un poco mejor esta diferencia.

Por otra parte, existen dos formatos equivalentes a IMC1 e IMC2, que son IMC3 e IMC4 respectivamente, salvo por el orden de las matrices U y V que se intercambia.

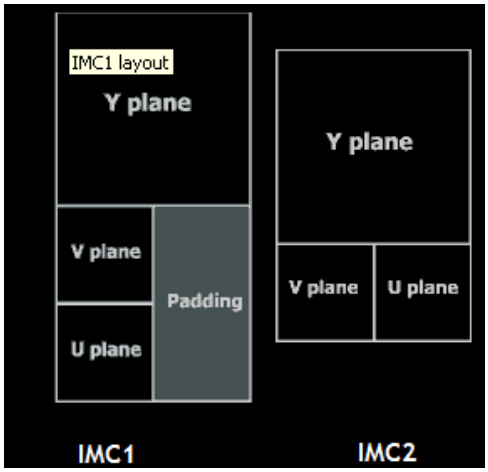


Figura 1.11 – Forma de almacenamiento de las matrices Y, U y V en IMC1 e IMC2.

1.3.3.2. Formatos 4:1:1

YVU9. En este caso tenemos un formato que reduce a un cuarto las muestras que se recogen para el color de la imagen. Se compone de una matriz de luminancia de dimensión N x N y una matriz para cada componente cromática de (N/4) x (N/4).

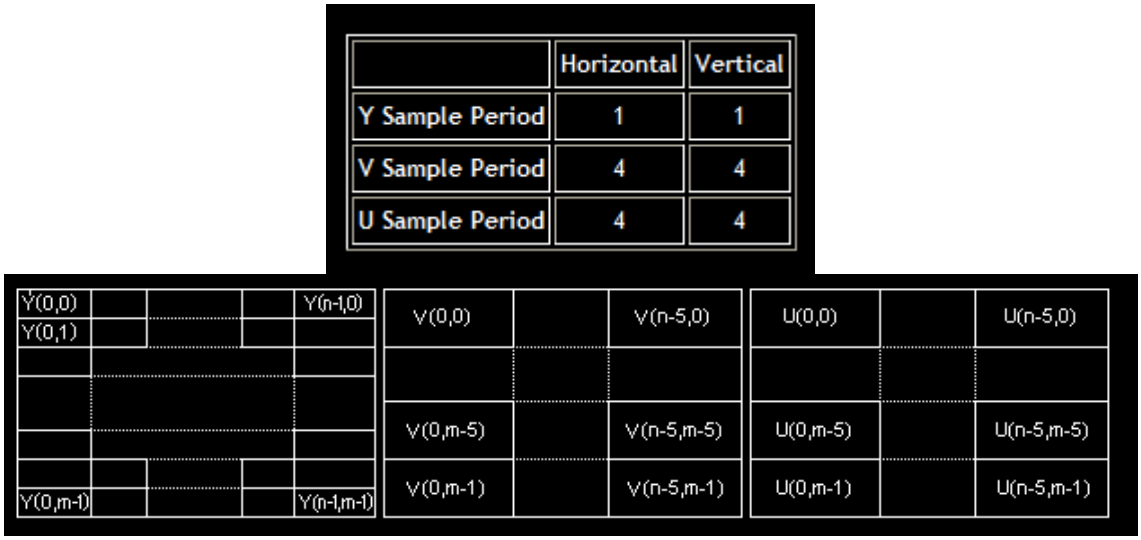


Figura 1.12 – Periodos de submuestreo y disposición de las matrices para YVU9.

1.3.3.3. Formatos 4:2:0

IYUV o I420. Si se busca información sobre este formato se puede comprobar que la información existente, es algo confusa, ya que por un lado se dice que es idéntico en las dimensiones de sus matrices a YV12 y por el otro, se habla de dos familias diferentes de formatos a las que se denominan 411 y 420 respectivamente, pero como ya hemos visto en el presente documento, YV12 es un formato 422 y no un 411, por ello esta información es tan confusa.

NV12 y NV21. Ambos formatos almacena en memoria todas las componentes Y en primer lugar en un array de caracteres sin signo y luego todas las componentes U y V intercaladas entre sí. La única diferencia radica en el orden en que se intercalan las componentes cromáticas, ya que el de NV21 es contrario al de NV12.

1.3.3.4. Otros formatos planos

YUV9. Es el formato utilizado por Intel para la tecnología Indeo Video. En este caso, se almacenan 16 bytes de luminancia por cada Byte de Crominancia, es decir, 128 bits de luminancia por cada 8 bits de crominancia.

Y800 e Y16. Estos dos formatos se utilizan para la generación de imágenes monocromáticas. En el primero, tan sólo se utilizan 8 bits para representar cada pixel, mientras que en el segundo se dobla esa cantidad para poder representar escalas de grises.

1.3.3.5. Diferencia entre los formatos 4:1:1 y 4:2:0

Una vez vistos los diferentes formatos YUV existentes que siguen una lógica de componentes separadas, se debe aclarar un concepto que en muchas ocasiones induce a error. Este concepto es la principal diferencia existente entre los formatos que siguen el esquema 411 y los que siguen el esquema 420. A priori, con lo visto hasta el momento se puede pensar que un formato 420 es similar a un 422, ya que según esa notación y lo visto hasta ahora, nos está indicando que una de las componentes cromáticas se desecha por completo, lo cual es incorrecto.

La diferencia entre un formato 411 y otro 420 radica principalmente en la forma de correspondencia entre componentes, esto es, la forma en la que se asocian las muestras de luminancia con las de crominancia a la hora de establecer una correspondencia entre ellas.

En el caso de los formatos 411, las muestras de luminancia están ordenadas por filas al igual que las de crominancia en sus respectivas matrices, de tal modo que cada cuatro muestras de luminancia de una misma fila corresponden a una muestra de cada componente de crominancia.

En el caso de un formato 420, cambia la relación de correspondencia, ya que las muestras de luminancia están agrupadas en matrices 2 x 2. Por lo tanto, en los formatos 420 la correspondencia es una matriz 2 x 2 por cada muestra que se toma de las componentes de crominancia. A modo de ejemplo, supongamos que tenemos una imagen de 16 pixeles. La estructura que seguiría cada formato sería.

Formato 4:1:1

Matriz Y				Matriz U		Matriz V	
Y0	Y1	Y2	Y3	U0	U4	V0	V4
Y4	Y5	Y6	Y7				
Y8	Y9	Y10	Y11	U8	U12	V8	V12
Y12	Y13	Y14	Y15				

Formato 4:2:0

Matriz Y				Matriz U		Matriz V	
Y0	Y1	Y4	Y5	U0	U4	V0	V4
Y2	Y3	Y6	Y7				
Y8	Y9	Y12	Y13	U8	U12	V8	V12
Y10	Y11	Y14	Y15				

Tabla 1.1 – Diferencia entre formato 4:1:1 y 4:2:0.

1.4. Multiplexado mediante algoritmos MDC

El MDC divide el video que se va a transmitir en capas denominadas descriptores. Estos descriptores contienen por separado, información suficiente para que, recibiendo tan sólo uno de ellos se pueda visualizar el video, pero con una calidad menor a la del original. Por tanto, si recibimos únicamente uno de los descriptores en los que se ha fragmentado el video, obtenemos la peor calidad del mismo y si los podemos recibir todos obtendremos la calidad del video original. En la Figura 1.13 se puede apreciar el proceso explicado a modo de ejemplo.

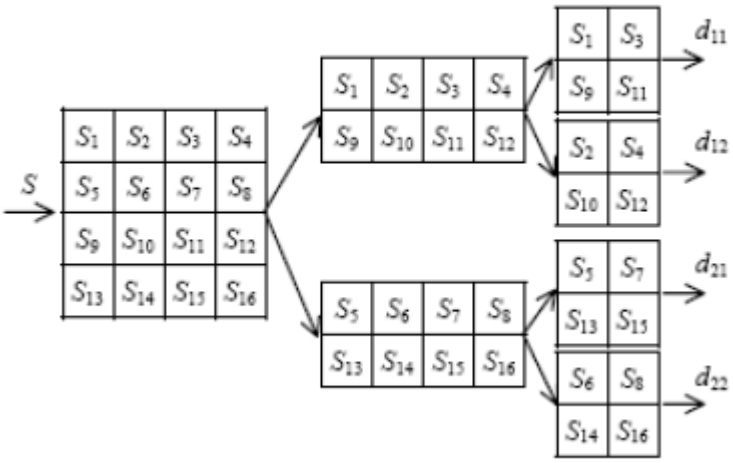


Figura 1.13 – Proceso de obtención de los descriptores.

Existen dos formas de implementar MDC. Mediante Escalabilidad espacial, o bien, mediante Escalabilidad Temporal.

1.4.1. Escalabilidad espacial

Esta técnica consiste en dividir el video original en una serie de capas hasta llegar a una capa que se denomina Base. Con esta capa base, podemos visualizar el video con la mínima calidad, siendo el resto de las capas las que aumentarán el número de detalles de la imagen, aumentando así la calidad del visionado hasta llegar a la del original una vez hemos sumado todas las capas en las que previamente se dividió. En la Figura 1.14 se puede ver un esquema del proceso mediante el cual se suman las diferentes capas en las que se divide el video.

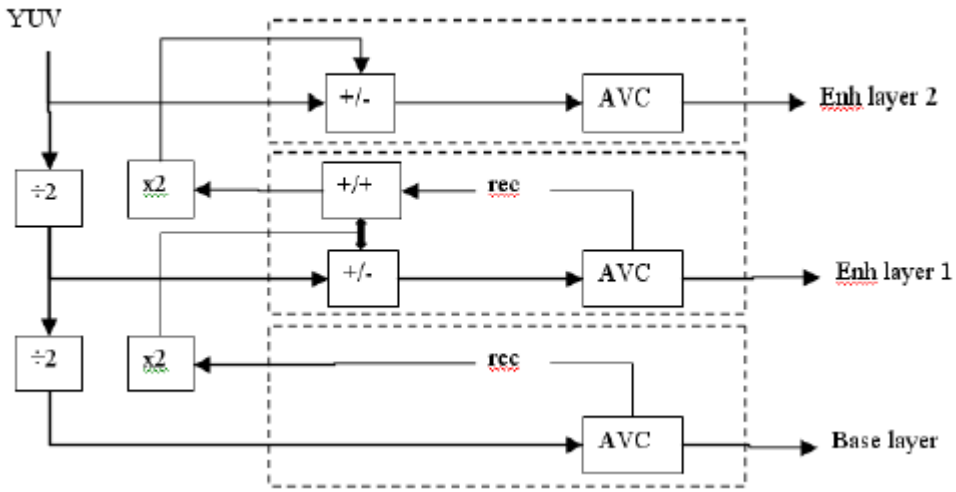


Figura 1.14 – Proceso de Multiplexación Espacial.

1.4.2. Escalabilidad temporal

Esta es otra forma para abordar la implementación de MDC. La principal diferencia con la técnica anterior es que en lugar de dividir la imagen en capas reduce el número de fotogramas por segundo que se envían al cliente, consiguiendo así que se pueda ver un video con un menor ancho de banda. Los fotogramas que faltan son reconstruidos por el receptor mediante técnicas de predicción o mediante un Filtro de Compensación temporal de movimiento. Esta última solución para recuperar los fotogramas en el receptor está actualmente en desuso, debido a su elevada complejidad. En la Figura 1.15, podemos observar cómo funciona el proceso de eliminación de los fotogramas.

Figura 1.15 – Proceso de Escalabilidad Temporal.

Capítulo 2
Desarrollo teórico

2.1. Introducción

Hasta ahora se han explicado las bases sobre las que se sustentará este proyecto, tales como el formato YUV y la codificación mediante múltiples descriptores (Multiple Description coding). En esta sección se explicarán los fundamentos teóricos en los que se basa el proyecto, atendiendo a las líneas más generales sin hacer referencia a ninguna herramienta o lenguaje específico.

Como ya se citó en la introducción de este documento, se pretende desarrollar una aplicación basada en el uso de la tecnología Streaming, que nos permite transmitir video en tiempo real. A continuación se comentarán los elementos de los que consta la aplicación a desarrollar desde un punto de vista genérico, explicándose con más detalle en las secciones posteriores.

2.2. Desarrollo de una aplicación con tecnología streaming

La idea se basa en desarrollar una aplicación cliente – servidor que permita el intercambio de video en formato YUV en tiempo real independientemente del ancho de banda disponible, siempre que este sea suficiente para transmitir la mínima unidad de video reproducible, utilizando para ello los algoritmos MDC para la división y ensamblado del video en partes más pequeñas que serán convertidas a otro formato diferente que requiera un menor ancho de banda para su transmisión.

La aplicación consta de los siguientes elementos:

Servidor. Almacena los videos que posteriormente se transmitirán por la red en función de la demanda de los clientes y envía los parámetros necesarios para que el sistema receptor tenga la suficiente información para llevar a cabo la reconstrucción de aquel que haya solicitado.

Sistema divisor. Utilizará los algoritmos MDC como base para preparar el video para la transmisión por la red, dividiéndolo en las unidades mínimas reproducibles para que en caso de que el cliente no sea capaz de recibir toda la información tenga un video de menor calidad pero igualmente funcional.

Códec. Permite convertir las diferentes partes del video a un formato que necesite menos ancho de banda que el YUV, para asegurar en la medida de lo posible, que el video llega siempre con la máxima calidad al receptor. Además este sistema de conversión estará presente tanto en la parte transmisora como en la receptora, desempeñando diferentes funciones dependiendo de donde nos encontremos, es decir, tendremos un sistema de conversión, que desempeñará, como ya se citó antes, la función de conversión de YUV a otro formato para transmitirlo por la red, función que desempeñará en la parte transmisora. En la parte receptora también estará presente dicho sistema para desempeñar dos funciones; La primera será la de revertir la conversión realizada por el transmisor, y la segunda la de adecuar el video a un formato de algún códec conocido para poder reproducirlo.

Sistema ensamblador. Se encargará de realizar el proceso inverso al del transmisor para unir las partes recibidas y obtener el video original a partir de ellas.

Cliente. Se encargará de recibir los datos del usuario sobre el video que desea visualizar, enviándolos al servidor preparando el sistema para la recepción de datos y la reproducción del video solicitado.

En la Figura 2.1 se muestra un esquema que ilustra lo explicado de forma gráfica.

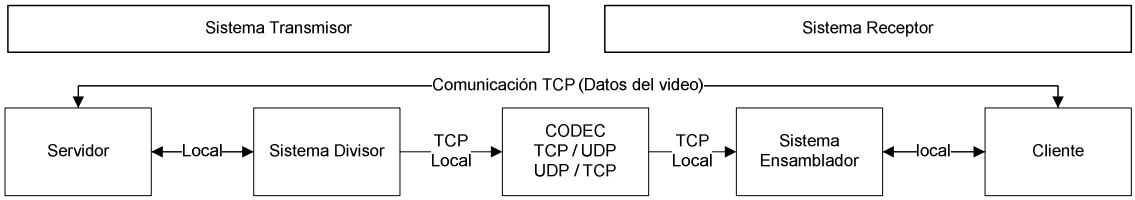


Figura 2.1 – Esquema general de la aplicación.

Otro punto a tratar a la hora de desarrollar una *aplicación de comunicaciones* como esta, es saber de qué tipo de aplicación estamos hablando, es decir, se debe conocer lo que *requiere* nuestra aplicación para utilizar unos protocolos u otros a la hora de transmitir la información. Si se trata de una aplicación donde la fiabilidad sea imprescindible en la transmisión, donde se deba asegurar un orden correcto en la llegada de los paquetes se utilizará TCP como protocolo de transporte. En cambio, si se trata de una aplicación donde tiene prioridad la velocidad de transmisión frente a lo demás, se utilizará UDP.

En nuestro caso se pretende desarrollar una aplicación que transmite video en Streaming, por lo que estamos hablando de una aplicación donde el tiempo es de máxima importancia, así que para las partes de nuestra aplicación que se encargan de la transmisión del video a través de la red utilizaremos el protocolo UDP, puesto que no necesitamos fiabilidad ni retransmisiones en la comunicación, sino velocidad.

Por otro lado, para las comunicaciones locales que lo necesiten, así como aquellas que existan entre cliente y servidor para enviar algún tipo de parámetro necesario para el correcto funcionamiento del sistema, utilizarán el protocolo TCP, ya que se necesita que la información llegue correctamente a todas las partes de la aplicación para asegurar que la división y el ensamblado del video se realizan con éxito.

A continuación se explicará con más detalle cada una de las partes de la aplicación tanto en el sistema transmisor como en el sistema receptor.

2.3. Esquema del sistema transmisor

Ahora que ya se tiene una visión general sobre la aplicación a desarrollar, se puede tomar como base para describir con más detalle cada una de sus partes, teniendo así una referencia más sólida y concreta, pero sin dar detalles sobre implementación o herramientas utilizadas para su desarrollo ya que esto se abordará en futuros apartados del presente documento.

Cada una de las partes que componen el sistema transmisor resuelve un problema asociado a ella, por lo tanto, a continuación se expondrán qué problemas han ido surgiendo a la hora de plantear el desarrollo de la aplicación y como se han solventado con cada una de las partes de las que se compone el sistema transmisor.

En primer lugar, el desarrollador de plantearse dos cuestiones fundamentales: los ***algoritmos MDC*** y el ***formato YUV***, que son la base de la aplicación.

Como ya vimos en el capítulo anterior, existían dos formas de implementar los algoritmos MDC que o bien se basaban en el tiempo o bien lo hacían en el espacio. Estas dos formas son lo que se conoce como ***Escalabilidad Temporal*** y ***Escalabilidad Espacial***. En el caso de la escalabilidad temporal, se

reducía la calidad eliminando fotogramas y reconstruyéndolos después en recepción, mientras que en el caso de la escalabilidad espacial, lo que se sacrificaba era la resolución dividiendo el video en capas que después se unirían en recepción para recuperar el original. En el caso concreto de esta aplicación se aplicará la técnica de Escalabilidad Espacial a la hora de trabajar con los algoritmos MDC. Existen a su vez, dos formas de abordar la escalabilidad espacial que se diferencian entre sí en la manera de tratar las líneas del video a la hora de dividirlo.

Los ***algoritmos MDC*** basados en escalabilidad espacial, pueden dividir el video en tantas capas como se desee siempre y cuando el número de capas sea múltiplo de dos y exista suficiente información como para poder reproducir el video a una resolución decente. Para explicar las formas y para la realización de los esquemas asumiremos que el video se divide en cuatro partes iguales, cada una de las cuales tiene la suficiente información para ser reproducida en recepción por sí sola.

Forma 1 o algoritmo MDC 1

La primera de las formas del algoritmo MDC, trata el video de forma distinta en función de si estamos en las líneas de luminancia (Y) o en las de crominancia (U ó V).

En el caso de las líneas de luminancia, los pixeles se asignan a una parte determinada extrayéndolos de dos líneas consecutivas, es decir, si nos centramos en la parte uno, el primer pixel de las dos primeras líneas de luminancia del video se asignaría a dicha parte. El tercero de ambas líneas también se asignaría a la parte uno, siguiendo una asignación del tipo 1, 3, 5, 7, … en el caso de los píxeles de una misma línea. Si nos fijamos en el conjunto de todas las líneas de luminancia, los píxeles se extraerían siguiendo una distribución del tipo 1, 2, 5, 6, 9, 10, …

Resumiendo, para los píxeles de luminancia cada uno de los patrones citados anteriormente a la hora de su extracción, corresponderían al ***submuestreo horizontal*** y al ***submuestreo vertical*** respectivamente.

En el caso de las muestras de crominancia, el submuestreo horizontal, así como el vertical, se realizan siguiendo el mismo patrón, el del tipo 1, 3, 5, 7, … lo que quiere decir que todas las líneas de crominancia son tratadas del mismo modo. En la Tabla 2.1 se muestra un esquema para el caso de una división en cuatro partes que ilustra a modo de esquema todo lo explicado anteriormente.

Componente de luminancia Y											
1	2	1	2	1	2	1	2	1	2	1	2
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
3	4	3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2	1	2
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
3	4	3	4	3	4	3	4	3	4	3	4
Componente de Crominancia U											
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
Componente de Crominancia V											
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4

Tabla 2.1 – Esquema de funcionamiento del algoritmo MDC 1.

Forma 2 o algoritmo MDC 2

La segunda forma, a diferencia de la primera, extrae los píxeles de video de la misma manera independientemente de cuál de las tres componentes esté tratando, es decir, que se trata con el mismo patrón de extracción a la luminancia y a la crominancia.

Si nos centramos al igual que en el caso anterior, en la primera de las cuatro partes, podemos ver que el patrón de extracción de los píxeles tanto para el submuestreo horizontal como para el submuestreo vertical es exactamente el mismo para cualquiera de las componentes siguiendo un patrón del tipo 1, 3, 5, 7, 9, …

En la Tabla 2.2 se muestra un ejemplo gráfico para una división en cuatro partes.

En lo referente al ***formato YUV*** nos surge un problema asociado a la transmisión del video, ya que se trata de *video sin compresión*, y su elevado tamaño requiere un *ancho de banda* demasiado alto para transmitirlo en tiempo real, por lo tanto se debe convertir a un formato alternativo. A este respecto, existen diversas opciones, siendo los más conocidos formatos como *mpeg* en sus distintas versiones, *divX* o *Xvid* entre otros. Una vez transformado el video a cualquiera de los formatos más comunes, ya podemos transmitirlo por la red sin problemas.

Componente de luminancia Y											
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
Componente de Crominancia U											
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4
Componente de Crominancia V											
1	2	1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4	3	4

Tabla 2.2 – Esquema de funcionamiento del algoritmo MDC 2.

Por otra parte, se deben conocer ciertos datos sobre el video sin los cuales no se podría llevar a cabo la división ni el ensamblado del mismo. Datos como el bitrate, los fps o la resolución son vitales para un correcto funcionamiento de los algoritmos MDC.

Una forma de obtener los datos sobre el video que necesitamos es a través de las cabeceras del formato, pero esta forma es completamente inservible para el formato YUV, ya que no tiene cabecera alguna y de sus datos en bruto no se puede extraer nada salvo la información del propio video.

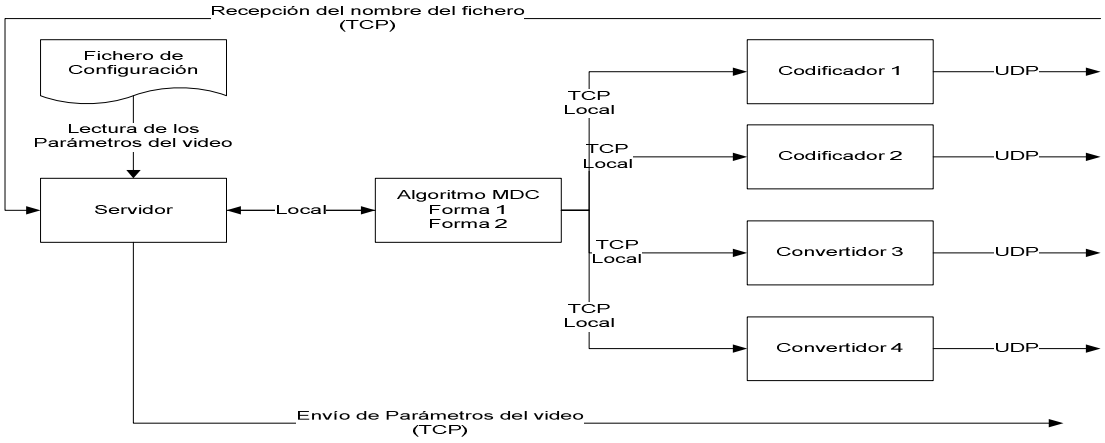


Figura 2.2 – Esquema del transmisor.

Otra forma de obtenerlos es mediante la creación de un fichero de configuración que permita al administrador del sistema añadir una lista de todos los videos con los que trabaja el sistema transmisor. De este modo el único requisito sería que la modificación del fichero sería manual y tendrían que añadirse parámetros nuevos por cada video adicional con el que el sistema tuviese que trabajar.

En la Figura 2.2 podemos ver un esquema de lo que sería en términos generales y teniendo en cuenta las premisas que se han explicado, el sistema transmisor.

2.4. Esquema del sistema receptor

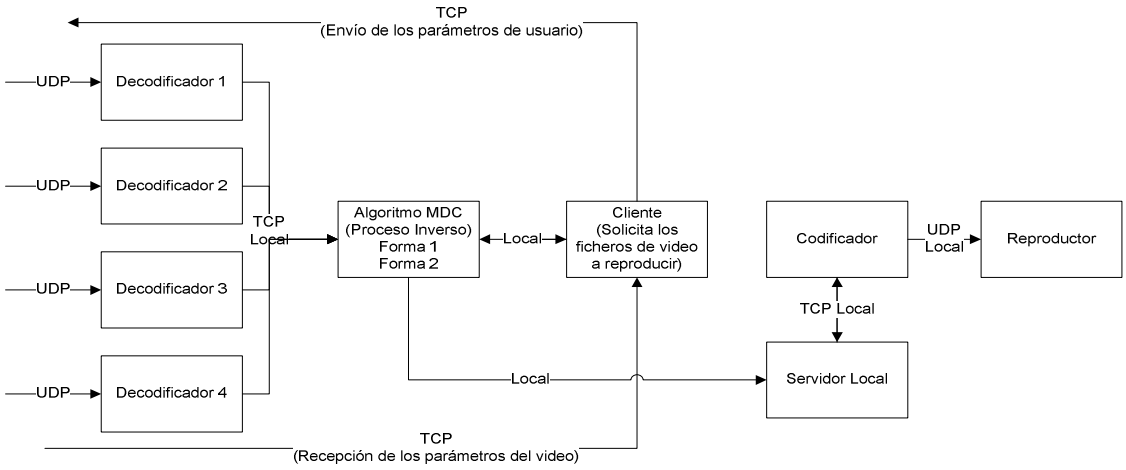


Figura 2.3 – Esquema del sistema receptor.

Una vez solucionados los problemas del transmisor, nos encontramos en la parte receptora con varios de estos mismos problemas, pero a la inversa. En el transmisor lo que se hacía era dividir el video original en partes más pequeñas y comprimirlas con alguno de los formatos más utilizados habitualmente como el mpeg (avi), el divX o el Xvid. Ahora lo que se ha de hacer es revertir lo que se hizo en transmisión, y por lo tanto debemos utilizar un decodificador para cada una de las partes codificadas en el formato comprimido que corresponda y un sistema ensamblador (al igual que el sistema divisor, su funcionamiento se basará en algoritmos MDC) que se encargará de unir las partes en las que se dividió el video en el sistema transmisor, como se puede ver en el esquema de la Figura 2.3.

Tras recuperar nuestro video solicitado en YUV, lo que debemos hacer es prepararlo para su reproducción en el programa que corresponda. Para dicha reproducción, se necesita realizar nuevamente una conversión a algún formato comprimido de los anteriormente citados para que el programa reproductor pueda procesar nuestro video. Para ello se dispone de un servidor local que se encarga de recibir el video del sistema ensamblador y de enviárselo al codificador para que este realice la conversión al formato comprimido más conveniente para el reproductor.

A modo de resumen, se podría decir, viendo la aplicación desde un punto de vista más global, que primero se divide y comprime el video y luego se descomprime cada una de sus partes y se ensambla, volviéndolo a comprimir para su reproducción, donde el receptor desempeñaría las tareas de decodificación, ensamblado y reproducción.

Capítulo 3
Implementación

3.1. Introducción

Este capítulo trata de abordar los diferentes problemas que se expusieron en el anterior, tratando las diferentes opciones disponibles para resolver dichos problemas y escogiendo la que es más acertada para lo que se debe implementar o la que más ventajas ofrezca en cuanto a determinados requisitos que se tienen en cuenta en la aplicación como pueden ser, manejo de la memoria, disponibilidad de herramientas de desarrollo o documentación, etc.

Por otra parte, una vez que se haya decidido sobre qué herramientas son las más adecuadas para el desarrollo de nuestra aplicación, se detallarán los diferentes módulos, atendiendo a las herramientas seleccionadas.

3.2. Lenguajes y herramientas

Ahora que ya conocemos qué tipo de aplicación queremos desarrollar y cuáles son los problemas que ésta nos propone solucionar, debemos escoger las diferentes herramientas con las que llevar a cabo su implementación.

La primera cuestión que nos debemos plantear es el lenguaje de programación a utilizar. Podemos utilizar un lenguaje que disponga de herramientas para la programación gratuitas y sea de código abierto como Java o C, o por el contrario, lenguajes como Visual .net que es un lenguaje corporativo y dispone de una serie de herramientas de pago para el desarrollo de aplicaciones.

En nuestro caso, para la realización de este proyecto, se ha optado por el lenguaje Java, debido a su amplio abanico de posibilidades en cuanto a las herramientas de desarrollo totalmente gratuitas y a su extensa documentación que permite tener una referencia muy útil para el desarrollador, gracias a su estructura ordenada por paquetes y clases específicas que dividen cada posible problema para que nos sea más fácil escoger un paquete o una clase concreta para abordar los problemas que se nos presenten.

Por otra parte debemos decidir qué herramienta será la que nos sirva de sistema de conversión que nos permita codificar el formato YUV a cualquiera de los formatos comprimidos más utilizados para la transmisión y reproducción del video en Streaming. Para este fin, se pueden escoger dos vías en función de hasta qué punto queramos controlar la codificación.

Si queremos un control total a bajo nivel de la codificación, se puede llevar a cabo la implementación de un programa que convierta el formato YUV a otro como por ejemplo mpeg, y otro que realice la función inversa, debido a que esto complicaría bastante el desarrollo del proyecto y provocaría un considerable retraso dada la complejidad de creación de sendas aplicaciones de conversión, esta opción ha sido descartada.

Otra opción es buscar una herramienta externa que nos permita realizar la conversión del formato YUV a cualquier otro y viceversa. Existen diversas opciones para la conversión entre formatos, pero generalmente son para conversiones entre formatos de uso común como mpeg, 3gp o mp4. Para el caso del formato YUV no existen demasiadas posibilidades, donde la mejor opción es el Proyecto FFMPEG (www.ffmpeg.org), que nos permite una conversión del formato YUV a diversos formatos de video comprimido y también su conversión inversa, por ello, serán utilizadas dos de las tres herramientas que

proporciona el Poryecto FFMPEG para el desarrollo de nuestra aplicación, utilizando mpeg como formato comprimido para la transmisión por la red.

Por último, debemos abordar la cuestión de la reproducción del video. Buscamos una herramienta que nos permita la reproducción de video en tiempo real (streaming), aunque en este caso, si que existen una gran variedad de herramientas donde elegir, como por ejemplo, Realplayer, Windows Media player, QuickTime o VLC entre otros.

Para el proyecto se utilizará VLC como reproductor, ya que FFMPEG, permite la comunicación por línea de comandos muy sencilla, además de una interfaz gráfica para el control de la reproducción con diversidad de opciones.

3.2.1. Java (Sun Microsystems)

Java es un lenguaje de programación orientado a objetos que nos permite modelar el mundo real a través de unas entidades que dan nombre a su paradigma, los objetos. Los objetos nos pueden servir para modelar el mundo real de forma muy representativa, como por ejemplo, un coche, una casa o una calle.
La entidad de Java que representa a un objeto es lo que se conoce como clase. Las clases son la base de todo programa Java, ya que representan con unas variables de estado llamadas variables ejemplares o de instancia, y unas funciones llamadas métodos las características y comportamiento de un objeto.

Estas líneas vendrían a ser un pequeño resumen de lo que principalmente nos permite Java como lenguaje, pero atendiendo a cuestiones más prácticas, lo que nos interesa saber, es por qué se ha elegido Java en lugar de otras opciones como podría ser C.

Las principales cuestiones por las que se suele elegir Java en lugar de otros lenguajes como C, suelen ser la gestión automática de memoria, la mejor abstracción pudiendo ver los diferentes módulos de la aplicación como objetos en lugar de cómo un conjunto de ficheros o la gran cantidad de documentación con la que cuenta su api. De hecho, además de lo anteriormente mencionado, el motivo principal por el que se ha escogido Java para este proyecto son las herramientas de desarrollo. Entornos de desarrollo muy completos que permiten al programador crear código de forma muy sencilla y corregir errores de manera muy dinámica, sobre todo aquellos asociados a la compilación o a la importación de librerías, además de contar con aplicaciones de depuración de programas para así poder realizar un mejor seguimiento de la ejecución de los mismos. Su documentación se puede encontrar en la página oficial de Sun Microsystems: http://java.sun.com/javase/reference/index.jsp.

3.2.2. El proyecto FFMPEG

FFMPEG es un rápido sistema de conversión de vídeo y audio, que además de permitir la conversión entre muchos de los formatos conocidos como MP4, MPEG para video y mp3 o amr para audio, entre otros, también nos permite realizar la conversión de YUV a MPEG, que es la que buscábamos.

FFServer nos permite transmitir video y audio que haya sido previamente convertido a alguno de los formatos soportados por FFMPEG además de hacerlo también en tiempo real combinándolo con FFMPEG, mediante flujos FFM. En nuestro caso, lo usaremos de esta última forma, ya que lo que nos interesa es transmitir vídeo en tiempo real.

FFPlay es un reproductor de medios muy sencillo que utiliza las librerías de FFMPEG y SDL. Normalmente, se utiliza como sistema de testeo cuando se trabaja con FFMPEG. Esta última herramienta no se utilizará en el proyecto. En su lugar se usará VLC.

Ahora que ya conocemos las distintas herramientas que vamos a utilizar, podemos empezar a analizarlas con un poco más de profundidad.

3.2.2.1. FFMPEG

Funcionamiento

FFMPEG, funciona por línea de comandos. La forma general de invocar a FFMPEG es la siguiente:

ffmpeg [[Opciones para el fichero de entrada][-i Fichero de entrada]]…{[Opciones para el fichero de salida] Fichero de salida}.

Mediante esta estructura general tan sencilla, FFMPEG no sólo nos permite convertir ficheros que se encuentren en nuestro PC, sino que podemos capturar el vídeo o el audio, en función de lo que nos interese, desde cualquier fuente, ya sea una tarjeta capturadora de video o una dirección IP de una máquina local o remota en la que se encuentre el fichero de audio o video a convertir.

A continuación veremos las opciones de FFMPEG más importantes y que se suelen usar con más frecuencia.

Opciones Principales

-h

Nos muestra la ayuda del programa, es decir, nos muestra cuales son las diferentes opciones que FFMPEG acepta como argumento.

-formats

Nos muestra una lista de los formatos que soporta el programa tanto de entrada como de salida. Los campos que preceden al nombre del formato nos dicen si ese formato es aceptado como parámetro de entrada o como parámetro de salida.

> *'D'* (*Decoding Available*). Si el nombre de un formato viene precedido de este campo, nos dice que es posible decodificar dicho formato, es decir, que FFMPEG lo puede recibir como parámetro de entrada.
>
> *'E'* (*Encoding available*). Este campo por su parte, no informa sobre la posibilidad de codificar este formato, lo que nos indica que se puede utilizar como parámetro de salida de FFMPEG.

-codecs

Esta opción nos permite saber cuales de los codecs (codificadores, decodificadores o ambos) están disponibles en el programa. Los campos que preceden al nombre de del códec, nos dicen si el codificador o el decodificador están disponibles y si lo están para audio o vídeo.

> *'D'* (*Decoding Available*). Nos indica si el decodificador está disponible para ese códec.
>
> *'E'* (*Encoding Available*). Si este campo está presente, nos dice que el codificador está disponible para el códec a cuyo nombre precede.
>
> *V/A/S*. Nos indica si el códec está disponible para audio, vídeo o subtítulos.

'S'. Es lo que nos indica si el códec soporta cortes.

'D'. El códec soporta renderizado directo.

'T'. El códec es capaz de manejar videos a la entrada que estén truncados aleatoriamente en lugar de en los límites de en los frames únicamente.

-f formato

La opción '-f ', nos permite especificar el formato que tiene nuestro video de entrada y el que tendrá el video a la salida.

-i

Esta opción es la que nos permite especificar la fuente de la que se extraerá el video, ya sea local o remota.

Si la fuente es local, tendremos sentencias como esta: ffmpeg –i video.yuv.
Si por el contrario, la fuente es remota podremos encontrarnos con sentencias como esta: ffmpeg –i tcp://192.168.3.25:5432.

-target type

Transforma el fichero de la entrada al formato que se indica en el campo 'type', ajustando automáticamente opciones como el bitrate, el códec, o el tamaño del buffer.

Opciones de Video

-b 'bitrate'

Nos permite modificar el ratio de bits (en Kbps) que se usará para nuestro video tanto a la entrada como a la salida.

-r 'fps'

Nos permite modificar el número de imágenes por segundo que se mostrarán en nuestro video.

-s 'size'

Con esta opción podemos establecer la resolución, ya sea manualmente introduciendo los valores de la forma 'WxH', donde 'W' es el ancho y 'H' la altura, o bien usando alguna de las abreviaturas que reconoce el programa para establecer aluna resolución estándar como por ejemplo HD a 1080p que se puede especificar como ' ffmpg –i video.avi –s hd1080 videohd.mpeg'.

-aspect 'aspect'

Incluyendo esta opción, podremos especificar el formato de pantalla en el que se va a visualizar el video, y por tanto lo adaptaremos a dicho formato de pantalla. Los más conocidos son el formato '4 : 3' y el formato '16 : 9' (pantalla panorámica).

-vcodec 'codec'

Nos permite establecer un códec determinado para realizar la conversión de video.

Opciones de Audio

-ar 'freq'

Con esta opción podemos seleccionar la frecuencia de muestreo para nuestro fichero de audio.

-ab 'bitrate'

Esto nos permite seleccionar el número de bits por segundo en el fichero de audio en Kilobits por segundo, donde sus valores más frecuentes son 64 Kbps, 128 Kbps o 192 Kbps.

-ac 'chanels'

Nos permite decidir a qué tipo de sonido se adaptará nuestro fichero de audio, mono (opción '1') o estéreo (opción '2').

Además de las citadas en este documento, FFMPEG, ofrece una amplia gama de opciones que nos permiten realizar múltiples cambios en los ficheros a la entrada y a la salida, ya sean de audio o de video, e independiente mente, de si su fuente es un fichero propiamente dicho o una tarjeta capturadora de televisión o una dispositivo externo de video digital como por ejemplo una cámara que grabe en formato dv1. Para tener una visión más completa sobre el funcionamiento de FFMPEG, conviene revisar la documentación oficial del proyecto que se puede encontrar en www.ffmpeg.org o bien, si ya tenemos instalado FFMPEG en el equipo, usar el comando 'man' para acceder a la ayuda del mismo.

Algunos Ejemplos de Aplicación

Ahora que ya hemos visto algunas de las opciones más importantes de FFMPEG de cara al proyecto, para una mejor comprensión nos valdremos de algunos ejemplos que servirán para tener una idea gráfica de lo que hacen las opciones y de cómo se pueden combinar.

Por ejemplo podemos convertir un fichero YUV420P a un .avi con la siguiente sentencia:

ffmpeg -i /home/prueba.yuv /home/salida.avi.

También podemos realizar el proceso contrario, obteniendo como resultado un fichero de video puro (YUV420P).

ffmpeg –i /home/video.avi videoPuro.yuv.

También podemos establecer más de un fichero a la entrada cuando queremos que estos se conviertan a un mismo formato de esta forma.

ffmpeg –i /tmp/a.wav –s 640x480 –i /tmp/a.yuv /tmp/a.mpg.

De esta forma convertiremos los dos ficheros tanto el de video como el de audio a mpeg. Tendremos dos ficheros nombrados exactamente igual, pero en el uno habrá audio y en el otro video.

Si se necesitan más ejemplos se puede acudir a www.ffmpeg.org y consultar la sección de documentación para tener todos los ejemplos de conversión entre formatos, ya sean de audio o de video.

3.2.2.2. FFServer

FFServer, como ya se dijo durante la presentación de los tres programas, es un sistema que nos permite transmitir ficheros de audio o video en tiempo real o con ficheros previamente convertidos. Este programa funciona mediante un fichero de configuración denominado ffserver.conf, situado en /etc. Para nuestro proyecto no será necesario modificar dicho fichero, ya que tomaremos la configuración por defecto.

Se puede ejecutar de forma individual utilizando la siguiente sintaxis:
ffserver [Opciones].

También podemos arrancarlo cuando ejecutamos FFMPEG solicitándole una fuente o un destino remoto.

A continuación veremos las principales opciones que FFServer nos proporciona para su ejecución por línea de comandos.

Opciones

-L

Nos muestra la licencia del programa.

-version

Nos indica la versión que tenemos instalada en nuestro equipo.

-formats

Nos muestra una lista de los formatos, codecs y protocolos que soporta FFServer, entre otros.

-f 'Fichero de Configuración'

Nos permite especificar un fichero de configuración alternativo al existente en /etc/ffserver.conf.

-n

Esta opción anula todas las directivas de ejecución que permiten al programa ejecutar FFMPEG de forma automática. De este modo, el usuario tendrá que ejecutar FFMPEG de forma manual.

3.2.3. El proyecto Videolan (VLC Media Player)

Al igual que en el caso de FFMPEG, se trata de un proyecto de código abierto y de distribución completamente gratuita. VLC es un reproductor con el que además de poder visionar casi cualquier medio, ya sea DVD, o video en formato comprimido nos permite la visualización de video a través de una URL dada o una dirección IP determinada, otorgándonos la posibilidad de ver video en Streaming, que es el que interesa para nuestro caso, indicando tan sólo el protocolo mediante el cual funciona el video que queremos ver y el nombre de la máquina, URL o dirección IP de la misma.

Por otra parte, frente a las otras opciones que se comentaban, VLC es la más adecuada, debido a que es la que menos recursos consume, ya que dispone de una sencilla y completa interfaz gráfica que nos permite controlar el programa de forma rápida y simple, haciendo mucho más fácil su uso tanto para el que visiona el video como para el desarrollador que la utiliza como reproductor.

En la página oficial del proyecto Videolan (http://www.videolan.org/vlc/) podemos encontrar un completo soporte tanto de documentación como de ayuda en sus foros, además de la descarga del programa para cualquiera de los sistemas operativos actuales más utilizados.

Resumiendo, VLC ha sido la aplicación elegida debido principalmente a su bajo coste en recursos de sistema, ya que tenemos que tener en cuenta que el reproductor se va a ejecutar en el cliente y debe de ser una aplicación que no cargue demasiado el sistema ya que esto excluiría a las máquinas más lentas, y su fácil utilización e instalación.

3.3. Módulos y clases de la aplicación

A continuación nos centraremos en los diferentes módulos de los que se compone nuestra aplicación y explicando cuál es el funcionamiento de cada uno de estos y sus distintos componentes.

3.3.1. El sistema transmisor

A continuación se va a abordar todos los detalles de implementación sobre el sistema transmisor que se vio en el desarrollo teórico agrupando las diferentes clases que podemos encontrar en el paquete Servidor de la aplicación en función del módulo al que representan.

3.3.1.1. Servidor

En este módulo se implementa las clases encargadas de aceptar a los clientes y de otorgarles servicio de forma concurrente según sea su demanda.

La Clase BufferBytes

public class **BufferBytes**
Se encarga de encapsular el tipo primitivo Byte para poder trabajar con él de forma dinámica y evitando así un consumo innecesario de recursos.

public **BufferBytes**(int tam)
Inicializa el array de Bytes con el tamaño especificado.
Parámetros: Tam. Define el tamaño que tendrá el buffer.

public void **AddByte**(byte b)
Añade un Byte al array en la última posición disponible.
Parámetros: b. Byte que se añadirá al array.

public void **InicializaPosicion**()
Devuelve a cero el contador que señala la última posición disponible para así poder sobrescribir el buffer y evitar una posible excepción por posición incorrecta (ArrayIndexOutOfBoundException).

La Clase RecolectorDatosTXT

public class **RecolectorDatosTXT**
Es la clase encargada de la extracción de los datos alojados en el fichero config.txt. Estos datos se encuentran en formato texto, por lo que necesitan ser convertidos al formato que corresponda para poder trabajar con ellos.

public **RecolectorDatosTXT**()
Crea el objeto para la obtención de datos del fichero config.txt.

public String[] **recolectorDatos**(String nombre_fichero) throws FileNotFoundException
Devuelve un array de objetos String que contiene los datos asociados al fichero con nombre nombre_fichero. En caso de no existir el nombre del fichero devuelve null
Parámetros: nombre_fichero. Nombre del fichero del que se debe obtener la información.

La Clase ConversorDatosVideo

public class **ConversorDatosVideo**
Se encarga de convertir los datos que se extraen del fichero config.txt para que sean accesibles para que se puedan utilizar en el resto de las clases.

public **ConversorDatosVideo**(String nombre) throws FileNotFoundException
Crea un Objeto que permite a las demás clases trabajar con los datos obtenidos.

public int **getAlto**()
Devuelve la altura de la pantalla en pixeles.

public int **getAncho**()
Devuelve la anchura de la pantalla en pixeles.

public double **getBitrate**()
Devuelve la tasa en bytes por segundo del video.

La Clase GestorServidores

public class **GestorServidores**
Se encarga de administrar todo lo relacionado con la concurrencia, creando las instancias necesarias para preparar la transmisión del video para cada uno de los nuevos clientes que entran al sistema.

public **GestorServidores**(int max_servidores)
Crea un nuevo gestor de servidores fijando el número máximo de clientes que se podrán conectar de forma concurrente al sistema.
Parámetros: max_servidores. Define el número máximo de clientes que se podrán conectar de forma simultánea al sistema.

public void **crearServidor**(DataInputStream dinfich, boolean algoritmo, AlgoritmosMDCDivisores divisor, String ipo, String ipd, int[] resolucion, DataOutputStream dout) throws InterruptedException, IOException
Crea las instancias necesarias para preparar los diferentes módulos asociados a un cliente determinado para transmitir el video.
Parámetros:
dinfich. - Es el flujo del que leeremos el video en el algoritmo MDC que corresponda.
algoritmo. Nos dice que versión del MDC se ejecutará. Si es true ejecutamos la primera, si es false ejecutamos la segunda.
divisor. Objeto que se encargará de la ejecución de la versión del algoritmo MDC que corresponda.
ipo. IP de nuestro servidor (será la usada por las instancias de ffmpeg como ip de origen).
ipd. IP del cliente conectado (será utilizada en las instancias de ffmpeg como ip de destino).
resolucion[]. Para fijar la resolución del video que procesará el algoritmo MDC que corresponda (Necesario para las instancias de ffmpeg).
dout. Flujo de salida para enviar los puertos de las instancias de ffmpeg al cliente que deberán situarse como puertos de escucha en éste para la decodificación.

La Clase ServidorFFMPEG1a4_9

public class **ServidorFFMPEG1a4_9** extends Thread
Servidor local encargado de la comunicación con los algoritmos MDC. Prepara los enlaces de comunicación necesarios Entre el algoritmo MDC que corresponda y las instancias de ffmpeg.

public **ServidorFFMPEG1a4_9**(DataInputStream ficherol, boolean decision, AlgoritmosMDCDivisores div, String ip, int[] enlaces)
Crea un servidor local inicializando sus parámetros (flujos de comunicación, forma del algoritmo MDC a ejecutar, etc).
Parámetros:
ficherol. Flujo del que se lee el fichero de video a tratar.
Decisión. Indica cuál de las dos formas del algoritmo MDC Divisor se ejecutará (true. Ejecuta la forma 1. False. Ejecuta la forma 2).
div. Objeto que contiene la implementación de las dos formas del algoritmo MDC Divisor.
Ip. Para asociar el servidor local a la misma ip que el resto del sistema.
Enlaces. Puertos que se asignarán a los flujos de comunicación.

public void **run**()
Ejecuta el hilo del servidor local.

La clase Servidor

public class **Servidor** extends Thread
Esta clase se encarga de aceptar las peticiones de los clientes y del envío de los datos sobre el fichero de video que estos solicitan para el correcto funcionamiento del algoritmo MDC en la reconstrucción. Es el sustento principal de la parte no gráfica del sistema transmisor.

public **Servidor**(boolean decision, int max_servidores, String ip, int port)
Crea un objeto servidor que se encarga se sitúa en escucha a la espera de las peticiones de los clientes. Necesita una interfaz de usuario para el paso de parámetros ya sea por línea de comandos o gráfica.
Parámetros:
decisión. Indica la forma del algoritmo MDC a ejecutar (Parámetro recogido de la interfaz de usuario para los algoritmos MDC).
max_servidores. Número máximo de servidores locales que se crearán de forma simultánea para atender a los clientes concurrentemente (Parámetro recogido de la interfaz de usuario para la gestión de la concurrencia).
ip. Dirección ip del servidor.
port . Puerto del servidor.

public void **run**()
Ejecuta el servidor manteniéndolo en espera para aceptar las peticiones de los sucesivos clientes.

3.3.1.2. Sistema divisor

En el sistema divisor se implementan todas aquellas clases que tienen relación con la obtención de los flujos de video de menor resolución para su posterior codificación a formato de video comprimido.

La clase AlgoritmosMDCDivisores

public class **AlgoritmosMDCDivisores**

Esta clase alberga todo lo referente a la división del video en flujos de baja resolución mediante las formas uno y dos del algoritmo MDC. Además, la clase deberá formar parte de otra que implemente Threads como base de su funcionamiento.

public **AlgoritmosMDCDivisores**(int[] resol, double br, int lf)
Crea un objeto que contiene todo lo necesario para trabajar con las formas uno o dos de los algoritmos MDC de división.
Parámetros:
resol[]. Resolución del video a dividir.
br. Tasa de bytes por segundo de video.
lf. Longitud total del fichero de video.

public void **algoritmoMDC_1**(DataInputStream dinl, DataOutputStream[] doutl) throws IOException, InterruptedException
Primera de las dos formas del algoritmo MDC. Este método divide el video que recibe a la entrada en cuatro flujos de menor resolución que enviará a traves de cuatro sockets que utili- zará como flujos de salida.
Parámetros:
dinl. - Es el flujo de entrada por el que le llegan los bytes del fichero. Puede ser un fichero local o un socket.
doutl[]. Contiene los cuatro flujos de salida que podrán ser tanto flujos referidos a rutas locales del sistema de archivos como a sockets (Principalmente el método está pensado para el envio mediante sockets).

public void **algoritmoMDC_2**(DataInputStream dinl, DataOutputStream[] doutl) throws IOException, InterruptedException
Este método implementa la segunda de las dos formas de realizar el algoritmo MDC. Al igual que en la forma uno, el método recibe como parámetros de entrada un flujo para la lectura del fichero y un array para el envío de los flujos de menor resolución, una vez el fichero ha sido dividido.
Parámetros:
dinl. Es el flujo de entrada por el que le llegan los bytes del fichero. Puede ser un fichero local o un socket.
doutl[]. Contiene los cuatro flujos de salida que podrán ser tanto flujos referidos a rutas locales del sistema de archivos como a sockets (Principalmente el método está pensado para el envio mediante sockets).

public int **getFPS**()
Devuelve el número de frames por segundo.

public int **getNumFrames**()
Devuelve el número total de frames del video.

public int[] **getResolucion**()
Devuelve la resolución del video.

3.3.1.3. Sistema de codificación

Estas clases tienen como fin convertir los diferentes flujos de baja resolución en formato YUV, a un formato de video comprimido para la transmisión por la red que será el mpeg.

La Clase Transmisor1

public class **Transmisor1** extends Thread

Esta clase se encarga de ejecutar la primera de las cuatro instancias de ffmpeg para la transmisión.

public **Transmisor1**(String n, String ipo, int po, String ipd, int pd, int[] resolucion)
Inicializa los parámetros necesarios para la ejecución de ffmpeg, que convertirá el video a mpeg y lo transmitirá por la red.
Parámetros:
n. Nombre del Thread (Parámetro heredado de la clase Thread para asignar un nombre al hilo que ejecuta ffmpeg).
ipo. Dirección ip que se establece como origen para ffmpeg.
po. Puerto que se establece como origen en la ejecución de ffmpeg
ipd. Dirección ip destino para ffmpeg.
pd. Puerto destino para ffmpeg.
Resolución[]. Resolución del video para ffmpeg.

public void **run**()
Ejecuta la instancia de ffmpeg y espera hasta que finalice.

La Clase Transmisor2

public class **Transmisor2** extends Thread
Esta clase se encarga de ejecutar la segunda de las cuatro instancias de ffmpeg para la transmisión.

public **Transmisor2**(String n, String ipo, int po, String ipd, int pd, int[] resolucion)
Inicializa los parámetros necesarios para la ejecución de ffmpeg, que convertirá el video a mpeg y lo transmitirá por la red.
Parámetros:
n. Nombre del Thread (Parámetro heredado de la clase Thread para asignar un nombre al hilo que ejecuta ffmpeg).
ipo. Dirección ip que se establece como origen para ffmpeg.
po. Puerto que se establece como origen en la ejecución de ffmpeg
ipd. Dirección ip destino para ffmpeg.
pd. Puerto destino para ffmpeg.
Resolución[]. Resolución del video para ffmpeg.

public void **run**()
Ejecuta la instancia de ffmpeg y espera hasta que finalice.

La Clase Transmisor3

public class **Transmisor3** extends Thread
Esta clase se encarga de ejecutar la tercera de las cuatro instancias de ffmpeg para la transmisión.

public **Transmisor3**(String n, String ipo, int po, String ipd, int pd, int[] resolucion)
Inicializa los parámetros necesarios para la ejecución de ffmpeg, que convertirá el video a mpeg y lo transmitirá por la red.
Parámetros:
n. Nombre del Thread (Parámetro heredado de la clase Thread para asignar un nombre al hilo que ejecuta ffmpeg).
ipo. Dirección ip que se establece como origen para ffmpeg.
po. Puerto que se establece como origen en la ejecución de ffmpeg
ipd. Dirección ip destino para ffmpeg.
pd. Puerto destino para ffmpeg.
Resolución[]. Resolución del video para ffmpeg.

public void **run**()

Ejecuta la instancia de ffmpeg y espera hasta que finalice.

La Clase Transmisor4

public class **Transmisor4** extends Thread
Esta clase se encarga de ejecutar la cuarta de las cuatro instancias de ffmpeg para la transmisión.

public **Transmisor4**(String n, String ipo, int po, String ipd, int pd, int[] resolucion)
Inicializa los parámetros necesarios para la ejecución de ffmpeg, que convertirá el video a mpeg y lo transmitirá por la red.
Parámetros:
n. Nombre del Thread (Parámetro heredado de la clase Thread para asignar un nombre al hilo que ejecuta ffmpeg).
ipo. Dirección ip que se establece como origen para ffmpeg.
po. Puerto que se establece como origen en la ejecución de ffmpeg
ipd. Dirección ip destino para ffmpeg.
pd. Puerto destino para ffmpeg.
Resolución[]. Resolución del video para ffmpeg.

public void **run**()
Ejecuta la instancia de ffmpeg y espera hasta que finalice.

3.3.1.4. La interfaz gráfica del sistema transmisor

En esta sección se agrupan todas las clases relacionadas con la interfaz de usuario. Como todas estas clases han sido generadas con el editor visual de interfaces gráficas que proporciona el entorno de desarrollo Netbeans, no se adjuntarán detalles sobre los métodos que contienen, debido a que todos ellos se han generado de forma automática con dicho editor visual de Netbeans, por lo tanto, se mostrará en su lugar una captura con la parte de la interfaz gráfica a la que corresponde cada clase haciendo así más sencilla su comprensión para el lector de este documento.

La Clase ServidorView

Esta clase contiene el frame principal de la interfaz gráfica. Sobre este frame se construyen el resto de componentes (Figura 3.1).

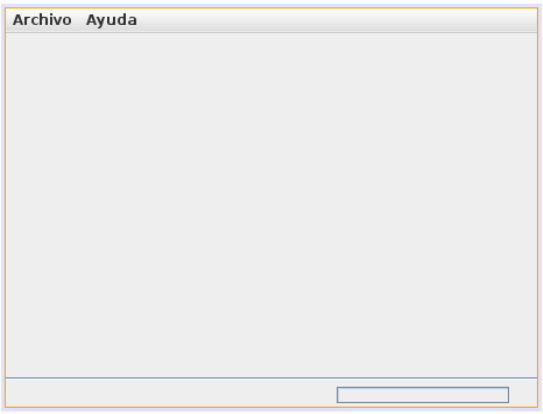


Figura 3.1 – Diseño de la ventana principal de la interfaz de usuario del Servidor (Ésta es la ventana sobre la que se ensambla el resto de la interfaz gráfica del servidor)..

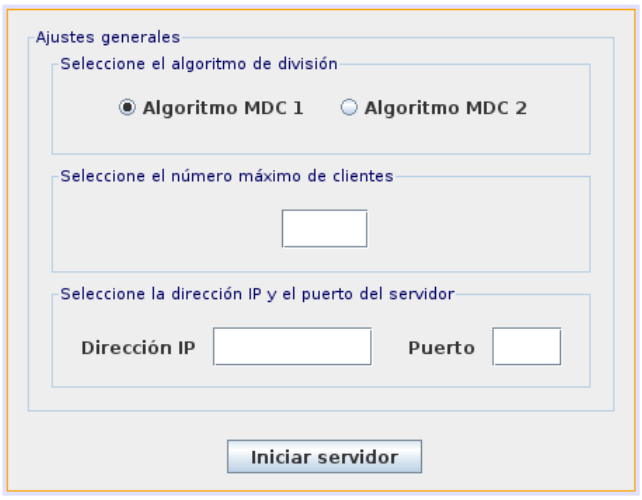


Figura 3.2 – Panel que contiene las opciones y los botones de interacción con el usuario.

La Clase ServidorGUI
Contiene el panel sobre el que se cargan todas las componentes del frame principal (Figura 3.2).

La Clase ServidorAboutBox
Es el cuadro de diálogo que muestra información sobre la aplicación como la versión del producto o el autor (Figura 3.3).

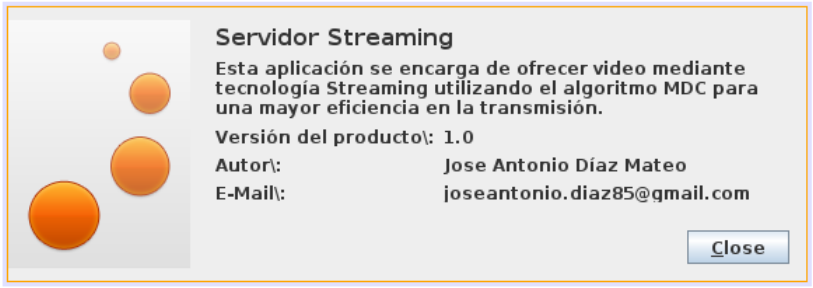


Figura 3.3 – Ventana de información sobre el servidor.

La Clase ServidorApp
Es la clase principal del sistema transmisor.

3.3.2. El sistema receptor

Hasta ahora hemos estado hablando sobre las clases del sistema transmisor y su implementación. En esta sección nos centraremos en las clases que podemos encontrar en el paquete cliente de nuestra aplicación que implementan las diferentes funcionalidades del sistema receptor.

3.3.2.1. Sistema de decodificación

El sistema de decodificación contiene el conjunto de clases necesario para revertir la conversión que se realizo en el sistema transmisor, por lo que la misión de este conjunto de clases se centra en la conversión de los diferentes flujos de video que le llegan de mpeg a YUV.

La Clase Receptor1

public class **Receptor1** extends Thread
Esta clase se encarga de ejecutar la primera de las cuatro instancias de ffmpeg para la recepción.

public **Receptor1**(String n, String ip_local_servidor, String ip_local_cliente, int puerto)
Crea un nuevo Receptor inicializando los parámetros necesarios para la ejecución de ffmpeg que realizará la conversión de mpeg a YUV de los datos recibidos.
Parámetros:
n. Heredada de la clase Thread. Nos permite cambiar el nombre que aparecerá para nuestro thread durante su ejecución.
ip_local_servidor. Dirección ip que la instancia de ffmpeg utiliza para recibir el video.
ip_local_cliente. Dirección ip local que utiliza ffmpeg para la comunicación con el sistema local de ensamblado.
puerto. El puerto de escucha que utiliza ffmpeg para recibir el video.

public void **run**()
Ejecuta la instancia de ffmpeg. Una vez ejecutado el ffmpeg se bloquea debido a que se recibe mediante udp y ffmpeg no puede determinar el fin de la transmisión. Deberá existir una clase que monitorice el estado de dicha transmisión para saber cuándo se deja de transmitir.

public static Process getProcess()
Devuelve una referencia a la instancia de ffmpeg que se ejecuta.

La Clase Receptor2

public class **Receptor2** extends Thread
Esta clase se encarga de ejecutar la segunda de las cuatro instancias de ffmpeg para la recepción.

public **Receptor2**(String n, String ip_local_servidor, String ip_local_cliente, int puerto)
Crea un nuevo Receptor inicializando los parámetros necesarios para la ejecución de ffmpeg que realizará la conversión de mpeg a YUV de los datos recibidos.
Parámetros:
n. Heredada de la clase Thread. Nos permite cambiar el nombre que aparecerá para nuestro thread durante su ejecución.
ip_local_servidor. Dirección ip que la instancia de ffmpeg utiliza para recibir el video.
ip_local_cliente. Dirección ip local que utiliza ffmpeg para la comunicación con el sistema local de ensamblado.
puerto. El puerto de escucha que utiliza ffmpeg para recibir el video.

public void **run**()
Ejecuta la instancia de ffmpeg. Una vez ejecutado el ffmpeg se bloquea debido a que se recibe mediante udp y ffmpeg no puede determinar el fin de la transmisión. Deberá existir una clase que monitorice el estado de dicha transmisión para saber cuándo se deja de transmitir.

public static Process getProcess()
Devuelve una referencia a la instancia de ffmpeg que se ejecuta.

La Clase Receptor3

public class **Receptor3** extends Thread
Esta clase se encarga de ejecutar la tercera de las cuatro instancias de ffmpeg para la recepción.

public **Receptor3**(String n, String ip_local_servidor, String ip_local_cliente, int puerto)
Crea un nuevo Receptor inicializando los parámetros necesarios para la ejecución de ffmpeg que realizará la conversión de mpeg a YUV de los datos recibidos.

Parámetros:
n. Heredada de la clase Thread. Nos permite cambiar el nombre que aparecerá para nuestro thread durante su ejecución.
ip_local_servidor. Dirección ip que la instancia de ffmpeg utiliza para recibir el video.
ip_local_cliente. Dirección ip local que utiliza ffmpeg para la comunicación con el sistema local de ensamblado.
puerto. El puerto de escucha que utiliza ffmpeg para recibir el video.

public void **run**()
Ejecuta la instancia de ffmpeg. Una vez ejecutado el ffmpeg se bloquea debido a que se recibe mediante udp y ffmpeg no puede determinar el fin de la transmisión. Deberá existir una clase que monitorice el estado de dicha transmisión para saber cuándo se deja de transmitir.

public static Process getProcess()
Devuelve una referencia a la instancia de ffmpeg que se ejecuta.

La Clase Receptor4

public class **Receptor4** extends Thread
Esta clase se encarga de ejecutar la cuarta de las cuatro instancias de ffmpeg para la recepción.

public **Receptor4**(String n, String ip_local_servidor, String ip_local_cliente, int puerto)
Crea un nuevo Receptor inicializando los parámetros necesarios para la ejecución de ffmpeg que realizará la conversión de mpeg a YUV de los datos recibidos.
Parámetros:
n. Heredada de la clase Thread. Nos permite cambiar el nombre que aparecerá para nuestro thread durante su ejecución.
ip_local_servidor. Dirección ip que la instancia de ffmpeg utiliza para recibir el video.
ip_local_cliente. Dirección ip local que utiliza ffmpeg para la comunicación con el sistema local de ensamblado.
puerto. El puerto de escucha que utiliza ffmpeg para recibir el video.

public void **run**()
Ejecuta la instancia de ffmpeg. Una vez ejecutado el ffmpeg se bloquea debido a que se recibe mediante udp y ffmpeg no puede determinar el fin de la transmisión. Deberá existir una clase que monitorice el estado de dicha transmisión para saber cuándo se deja de transmitir.

public static Process getProcess()
Devuelve una referencia a la instancia de ffmpeg que se ejecuta.

La Clase TimeOut

public class **TimeOut** extends TimerTask
Esta clase se encarga de detener las instancias receptoras de ffmpeg cuando se bloquean y ya no reciben más información según los designios de un temporizador.

public **TimeOut**()
Crea un objeto TimeOut que será controlado por el temporizador.

public void **run**()
Ejecuta el hilo controlado por el temporizador que permite la finalización de las instancias de ffmpeg en recepción.

3.3.2.2. El servidor local y el sistema ensamblador

A continuación se van a detallar las clases referentes al sistema ensamblador y el servidor local que se comunica con el reproductor para la visualización del video.

La Clase ServidorFFMPEGReceptor4a1_8

public class **ServidorFFMPEGReceptor4a1_8** extends Thread
Se encarga de la comunicación con el sistema reproductor así como de la recuperación del video original invirtiendo los algoritmos MDC ejecutados en el sistema transmisor en la forma que corresponda.

public ServidorFFMPEGReceptor4a1_8(boolean decision, int calidad_video, int[] datos_mdc)
Crea un nuevo servidor local que preparará las conexiones y puertos necesarios para la recepción del video y su ensamblado.
Parámetros:
decisión. Indica que forma del algoritmo MDC se ejecutará. true ejecuta la forma 1. false ejecuta la forma 2.
calidad. Define la calidad a la que se reproducirá el video. En función del valor introducido en este parámetro el video se reconstruirá con una cantidad mayor o menor de flujos. Si el valor es 1 la calidad de la reproducción será la mínima posible. Si el valor es 2 la calidad se ajustará a un valor intermedio. Por último, si el valor es 3, el video se reproducirá con la máxima calidad posible.
datos_mdc. Son los datos necesarios sobre el video para una correcta ejecución del algoritmo de reconstrucción (resolución, frames por segundo y número total de frames del video).

public void **run**()
Ejecuta el servidor local y las tareas de ensamblado del video.

public void **algoritmoMDC_1Inverso**(int[] resol, int fpseg, int nf, DataInputStream[] dinl, DataOutputStream doutl) throws IOException, java.lang.InterruptedException
Proceso inverso al algoritmo MDC 1 que reconstruye el video original a máxima calidad.
Parámetros:
resol[]. - Array que contiene la resolución del video a reconstruir.
fpseg. Tasa de frames por segundo del video.
nf. Cantidad total de frmaes del video a reconstruir.
dinl[]. Array que contiene los cuatro flujos de entrada que utiliza el método para leer los datos de los flujos de baja resolución.
doutl. Flujo de salida por el que se envía el video reconstruido para su reproducción (El video se envía en formato YUV. Deberá ser adaptado a algún formato que el reproductor de video pueda soportar).

public void **algoritmoMDC_1Inverso2a1**(int[] resol, int fpseg, int nf, DataInputStream[] dinl, DataOutputStream doutl) throws IOException, InterruptedException
Proceso inverso al algoritmo MDC 1 que reconstruye el video original con la mínima calidad.
Parámetros:
resol[]. - Array que contiene la resolución del video a reconstruir.
fpseg. Tasa de frames por segundo del video.
nf. Cantidad total de frmaes del video a reconstruir.
dinl[]. Array que contiene los cuatro flujos de entrada que utiliza el método para leer los datos de los flujos de baja resolución.
doutl. Flujo de salida por el que se envía el video reconstruido para su reproducción (El video se envía en formato YUV. Deberá ser adaptado a algún formato que el reproductor de video pueda soportar).

public void **algoritmoMDC_1Inverso3a1**(int[] resol, int fpseg, int nf, DataInputStream[] dinl, DataOutputStream doutl) throws IOException, InterruptedException
Proceso inverso al algoritmo MDC 1 que reconstruye el video con una calidad media.

Parámetros:
resol[]. - Array que contiene la resolución del video a reconstruir.
fpseg. Tasa de frames por segundo del video.
nf. Cantidad total de frmaes del video a reconstruir.
dinl[]. Array que contiene los cuatro flujos de entrada que utiliza el método para leer los datos de los flujos de baja resolución.
doutl. Flujo de salida por el que se envía el video reconstruido para su reproducción (El video se envía en formato YUV. Deberá ser adaptado a algún formato que el reproductor de video pueda soportar).

public void **algoritmoMDC_2Inverso**(int[] resol, int fpseg, int nf, DataInputStream[] dinl, DataOutputStream doutl) throws InterruptedException, IOException
Proceso inverso al algoritmo MDC 2 que reconstruye el video original con la máxima calidad.
Parámetros:
resol[]. - Array que contiene la resolución del video a reconstruir.
fpseg. Tasa en frames por segundo del video.
nf. Cantidad total de frmaes del video a reconstruir.
dinl[]. Array que contiene los cuatro flujos de entrada que utiliza el método para leer los datos de los flujos de baja resolución.
doutl. Flujo de salida por el que se envía el video reconstruido para su reproducción (El video se envía en formato YUV. Deberá ser adaptado a algún formato que el reproductor de video pueda soportar).

public void **algoritmoMDC_2Inverso2a1**(int[] resol, int fpseg, int nf, DataInputStream[] dinl, DataOutputStream doutl) throws IOException, InterruptedException
Proceso inverso al algoritmo MDC 2 que reconstruye el video original con la mínima calidad.
Parámetros:
resol[]. - Array que contiene la resolución del video a reconstruir.
fpseg. Tasa en frames por segundo del video.
nf. Cantidad total de frmaes del video a reconstruir.
dinl[]. Array que contiene los cuatro flujos de entrada que utiliza el método para leer los datos de los flujos de baja resolución.
doutl. Flujo de salida por el que se envía el video reconstruido para su reproducción (El video se envía en formato YUV. Deberá ser adaptado a algún formato que el reproductor de video pueda soportar).

public void **algoritmoMDC_2Inverso3a1**(int[] resol, int fpseg, int nf, DataInputStream[] dinl, DataOutputStream doutl) throws IOException, InterruptedException
Proceso inverso al algoritmo MDC 2 que reconstruye el video original con la calidad intermedia.
Parámetros:
resol[]. - Array que contiene la resolución del video a reconstruir.
fpseg. Tasa en frames por segundo del video.
nf. Cantidad total de frmaes del video a reconstruir.
dinl[]. Array que contiene los cuatro flujos de entrada que utiliza el método para leer los datos de los flujos de baja resolución.
doutl. Flujo de salida por el que se envía el video reconstruido para su reproducción (El video se envía en formato YUV. Deberá ser adaptado a algún formato que el reproductor de video pueda soportar).

3.3.2.3. El reproductor

La reproducción se lleva a cabo una vez el video ha sido convertido a mpeg. A continuación veremos las clases que desempeñan el cometido de convertir el video a mpeg una vez este ha sido reconstruido por el sistema ensamblador, y de reproducirlo.

La Clase Intermediario

public class **Intermediario** extends Thread
El intermediario tiene como misión recoger el video ya ensamblado del servidor local en formato YUV, y convertirlo a mpeg para su reproducción.

public **Intermediario**(String n, int[] resolucion, String ip_local)
Crea una instancia del intermediario inicializando los parámetros que necesita la instancia de ffmpeg que este ejecuta para su correcto funcionamiento.
Parámetros:
n. Nombre que se le atribuye al hilo en ejecución (heredado de la clase Thread).
resolución[]. Resolución del video a convertir.
ip_local. Dirección ip de la máquina en la que se ejecuta el programa. Es la ip del servidor local.

public void **run**()
Ejecuta la instancia de ffmpeg que se encarga de la conversión del video y su transmisión al reproductor.

La Clase Reproductor

public class **Reproductor** extends Thread
Es el hilo encargado de la ejecución de VLC.

public void **run**()
Ejecuta el reproductor VLC y lo mantiene a la espera para recibir el video.

3.3.2.4. La interfaz gráfica del sistema receptor

Al igual que en el sistema transmisor, en esta sección se expondrán las diferentes clases que componen la interfaz gráfica del sistema receptor, teniendo en cuenta que se trata de código autogenerado por Netbeans, por lo que se expondrá un captura de la parte de al interfaz gráfica que representa la clase y no sus métodos.

La Clase ClienteView

Contiene el frame principal de la interfaz gráfica del sistema transmisor. Sobre este frame se montan el resto de componentes de la parte gráfica (Figura 3.4).

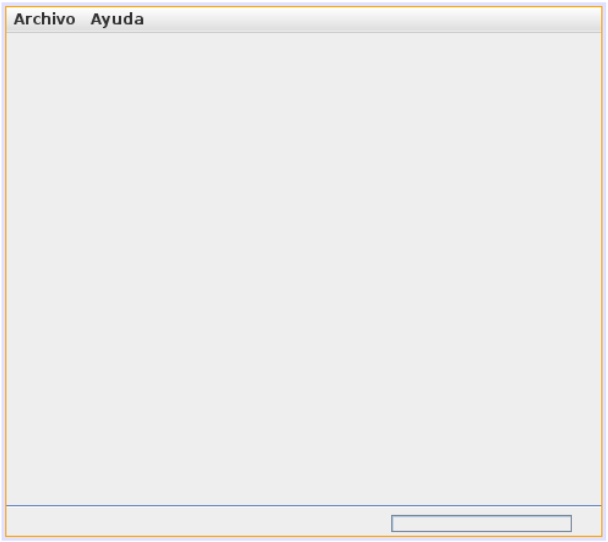


Figura 3.4 – Diseño de la ventana principal del cliente. Este frame contiene el resto de la interfaz gráfica.

La Clase ClienteGUI

Contiene el panel que alberga todos los controles de la interfaz gráfica (Figura 3.5).

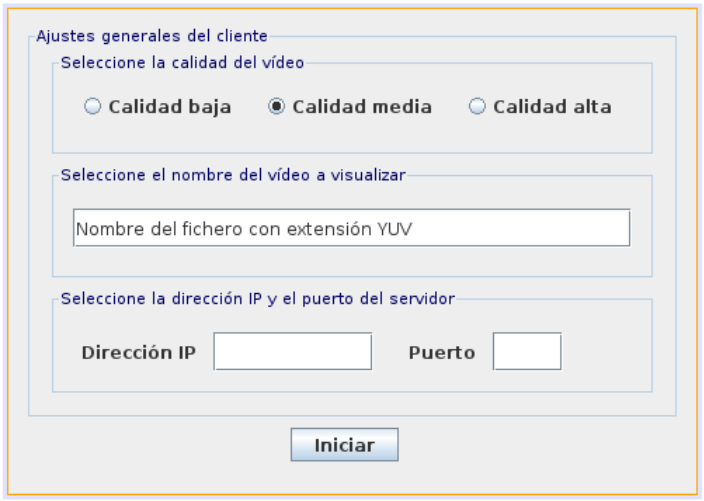


Figura 3.5 – Panel principal del cliente. Contiene todos los botones y opciones para la interacción con el usuario.

La Clase ClienteAboutBox

Contiene un cuadro de diálogo que muestra información sobre la versión del producto o el autor (Figura 3.6).

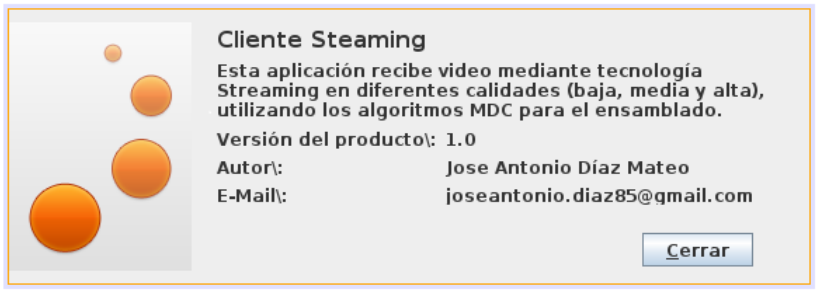


Figura 3.6 – Ventana de información sobre el cliente.

La Clase ClienteApp

Es la clase principal del sistema receptor.

Capítulo 4
Funcionamiento

4.1. Introducción

Hasta el momento se ha tratado lo referente a la forma teórico-práctica de llevar a cabo el desarrollo de una aplicación que nos permitiera visualizar video en Streaming. Hemos visto un planteamiento teórico que definía los diferentes módulos de los que constaba el sistema, así como su implementación. En el presente capítulo se abordará el funcionamiento de la interfaz de usuario de las aplicaciones cliente y servidor, así como del fichero config.txt.

En el fichero zip denominado aplicación, encontramos dos carpetas que corresponden a cliente y servidor. En el cliente encontramos lo siguiente: El fichero cliente.jar es el fichero que nos permite ejecutar la aplicación cliente. Para poder ejecutarlo deberemos utilizar el programa Java Runtime o un terminal. La carpeta lib. En esta carpeta nos encontramos con las librerías generadas por Netbeans para la ejecución de la interfaz gráfica.

En el caso del servidor nos encontramos con lo siguiente: El fichero servidor.jar. Este es el fichero que nos permite ejecutar el servidor. En este caso se ejecuta de la misma forma que en el anterior, bien por JRE o mediante un terminal. La carpeta lib. Al igual que en el cliente, es la carpeta que contiene las librerías generadas por Netbeans para el correcto funcionamiento de la interfaz gráfica. El fichero config.txt. El fichero de configuración que contiene a información sobre los ficheros de video que se alojan en el servidor.

4.2. La interfaz de usuario del servidor

Cuando arrancamos el servidor aparece una ventana como la de la Figura 4.1.

Figura 4.1 – Ventana principal del servidor.

Como se puede ver, tenemos la posibilidad de seleccionar el algoritmo MDC que utilizará el servidor para la división del video. Por defecto, aparece el algoritmo MDC 1, ejecutándose en caso de que no variemos la selección. Además, también podemos ajustar parámetros como el número máximo de

clientes que se podrán conectar de forma simultánea a nuestro servidor, así como la dirección IP y el puerto con el que se asociará.

Selección del Algoritmo MDC a Ejecutar

Esta opción nos permite decidir cuál de las dos formas del MDC se ejecutará. Una vez seleccionada no podrá ser modificada durante la ejecución del servidor, teniendo que reiniciar éste en caso de que se tenga que modificar.

Selección del Número Máximo de Clientes

Esta opción nos permite ajustar el número máximo de clientes que se podrán conectar a nuestro servidor de manera simultánea.

Se debe tener en cuenta que a mayor número de clientes conectados, mayor será la carga en el sistema y la utilización de recursos, puesto que la JVM tendrá que gestionar mayor número de hilos, dando lugar a posibles sobrecargas del procesador en caso de que el número sea excesivamente alto, produciendo una posible degradación en el servicio a los clientes.

Dirección IP y Puerto

Con estos dos campos, se pueden seleccionar la IP y el puerto de escucha de nuestro servidor, los cuales deberán ser conocidos para que se puedan conectar los clientes.

Figura 4.2 – Diálogo de parámetros incorrectos del servidor.

Cuando pulsamos sobre el botón de arranque "*Iniciar Servidor*", el sistema comprueba si todos los parámetros se han introducido correctamente. En caso contrario, la interfaz gráfica nos avisa con un cuadro de diálogo que indica el parámetro o parámetros que faltan por introducir, como se observa en la figura.

Una vez iniciado, el botón "Iniciar Servidor" pasa a un estado de inaccesibilidad para evitar el inicio de múltiples instancias. Para finalizar el servidor debemos pulsar el botón cerrar de la ventana de la interfaz gráfica o el botón salir del menú archivo.

Figura 4.3 – Servidor iniciado y escuchando.

Los Menús

Disponemos de dos menús: Archivo y Ayuda. En el menú "Archivo" se aloja el botón "Salir" que permite cerrar la aplicación, mientras que en el de "Ayuda" tenemos el botón Acerca de… que nos muestra la información sobre la aplicación.

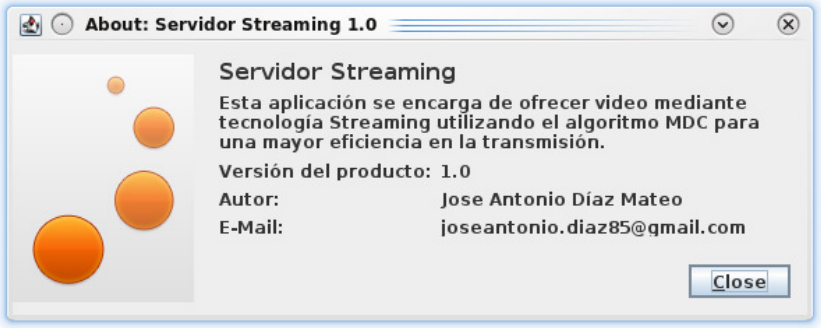


Figura 4.4 – Ventana de información del servidor.

4.2.1. El fichero config.txt

El fichero config.txt es la base de datos sobre los ficheros de video que pueden ser visualizados por los clientes. En la figura podemos ver un ejemplo del fichero config.txt que nos muestra cuál es la estructura que sigue para diferenciar a los videos que se almacenan en el servidor.

Los Campos de un Video en el Fichero config.txt

Cada video tiene su nombre y los campos referentes a la resolución del video y su tasa en Bytes por segundo que son los parámetros que necesita el servidor para poder dividir el video con los algoritmos MDC. Todo lo que no cumpla el formato establecido tal como se expone en el ejemplo provocará un mal funcionamiento del servidor.

Por ejemplo:

Hora Punta 3.yuv (El fichero de video con dicho nombre deberá encontrarse en el mismo lugar que el ejecutable servidor.jar). /home/carpeta personal/videos/Hora Punta 3.yuv (El fichero será localizado en la ruta especificada). A continuación se dará una breve descripción de lo que debe contener cada campo siguiendo el formato del ejemplo.

Nombre del fichero

El nombre del fichero debe situarse tal como en el ejemplo, sin espacio después del símbolo "=" o después de la extensión del mismo. Por ejemplo: La siguiente sintaxis sería incorrecta. +Nombre del Fichero= video de prueba.yuv. En cambio, la siguiente sintaxis sí sería correcta. +Nombre del Fichero=video de prueba.yuv. El símbolo "." no se deberá incluir en un fichero de configuración real, pues sólo es una forma de hacer más fácil la comprensión del ejemplo.

Resolución (Ancho y Alto de la pantalla) y Bitrate

En lo referente a los campos, la forma de agregarlos es idéntica a la del nombre cambiando únicamente el símbolo inicial para así diferenciarlos del nombre del fichero, sustituyendo el símbolo "+" por un símbolo "*". Un ejemplo de campo sería el siguiente:

Campo introducido correctamente. *Ancho de pantalla=176.
Campo introducido de forma incorrecta. *Ancho de pantalla= 176 .

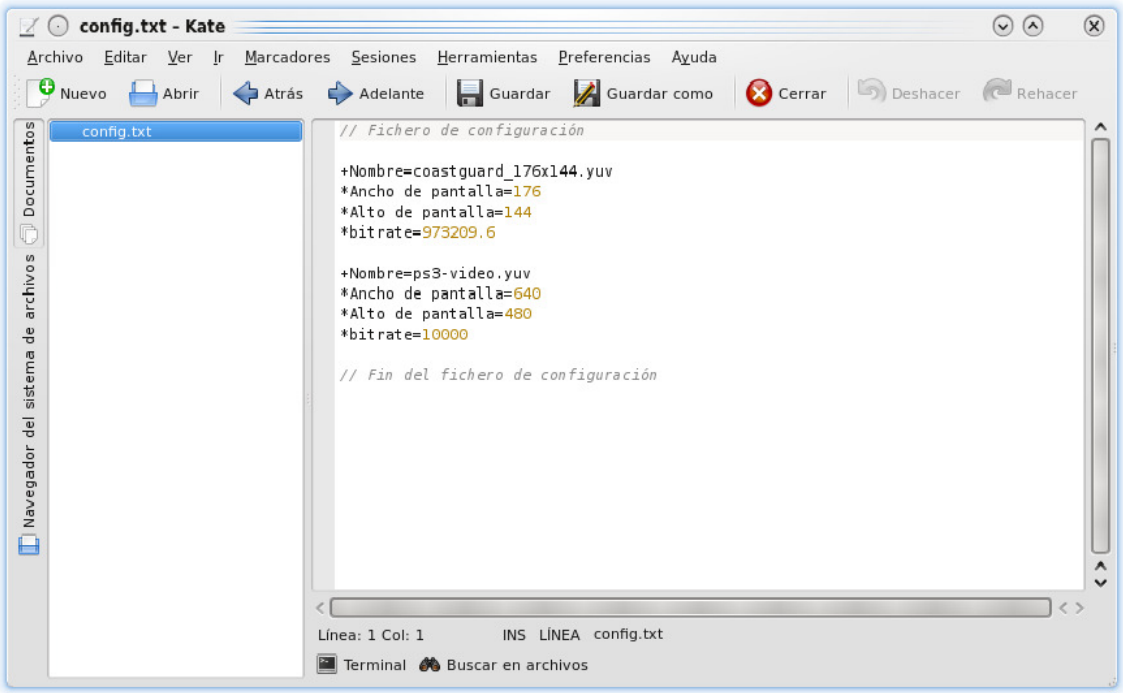


Figura 4.5 – Fichero de configuración del servidor.

4.3. La interfaz de usuario del cliente

En el caso del cliente, tenemos una interfaz similar, con una serie de opciones para determinar la calidad a la que queremos visualizar el video, el nombre del video en cuestión, o la dirección IP y el puerto a los que nos debemos conectar, como se puede ver en la figura.

Calidad de Video

En el cliente disponemos de tres posibles calidades de video. Esto influirá notablemente en la reproducción del video, ya que a menor calidad, la imagen se visualizará más degradada.

Calidad Baja. Es la peor de las tres, y aunque consume menos ancho de banda, la reproducción es bastante deficiente.

Calidad Media. En este caso, también se percibe un degradado de la imagen en la visualización, pero considerablemente menor.

Calidad Alta. Si seleccionamos esta opción estaremos pidiendo al cliente que la reconstrucción sea completa, permitiendo así que el video se vea con la resolución original y sin degradar la imagen. Esta opción ofrece muy buenos resultados en cuanto al visionado, pero requiere más recursos.
Se debe tener en cuenta que la calidad final del video que recibe el cliente no depende únicamente de la opción seleccionada, sino también de la calidad del video original y del algoritmo MDC con el que se haya arrancado el servidor.

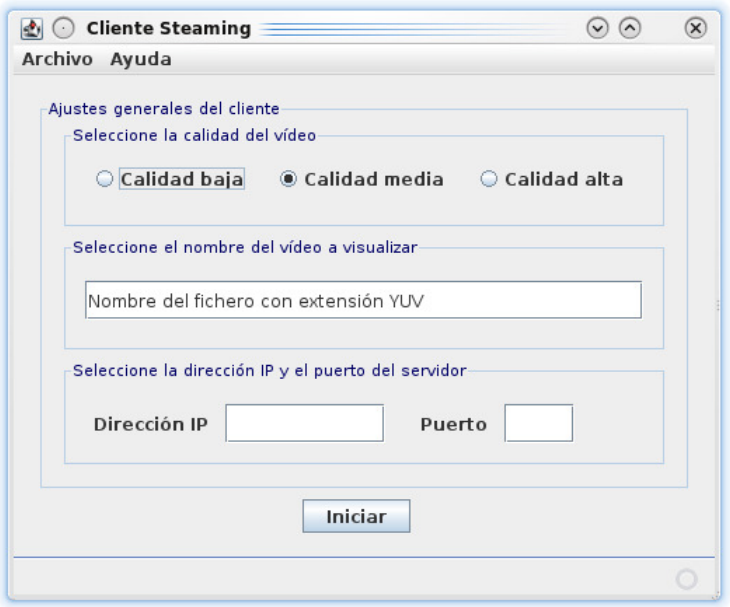


Figura 4.6 – Ventana principal del cliente.

Figura 4.7 – Diálogo de parámetros incorrectos del cliente.

Nombre del Fichero de Video

Esta opción nos permite decidir el video que se visualizará. Para saber cuál es el nombre exacto del fichero de video que hay que introducir en este campo, tendremos que disponer de una lista pública o sitio web donde se expongan los diferentes ficheros de video que se ofrecen en caso de que se pretendiera lanzar como un servicio, o simplemente dar a los clientes una lista de los videos que se ofrecen por algún otro medio, dependiendo del ámbito en el que queramos lanzar la aplicación.

Dirección IP y Puerto

Estos dos campos son los que nos permitirán conectarnos al servidor para recibir el video, por lo que deberán ser conocidos para que todos los usuarios puedan acceder a él. Por otra parte, también disponemos de un sistema de alertas (Figura 4.7) para el caso de que algún parámetro no se haya introducido correctamente que lanza un mensaje en caso de que falte algún parámetro o no sea correcto para evitar un mal funcionamiento del sistema. Una vez hemos definido correctamente los parámetros sobre el video y el servidor al que nos conectamos, pulsamos sobre el botón “Iniciar” y esperamos unos segundos a que el sistema se prepare para la reproducción, apareciendo la ventana del reproductor VLC.

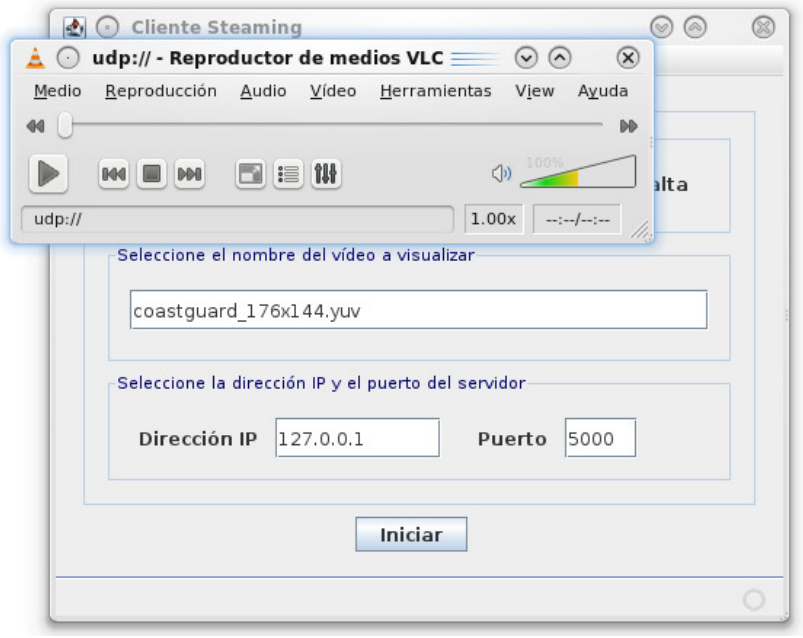


Figura 4.8 – Cliente esperando datos de vídeo.

Tras unos segundos comenzará la reproducción del video seleccionado. Cuando haya finalizado la reproducción del video completo podremos volverlo a reproducir o seleccionar otro video diferente. Además podremos conectarnos a otro servidor de Streaming que siga el mismo funcionamiento para el tratamiento del video.

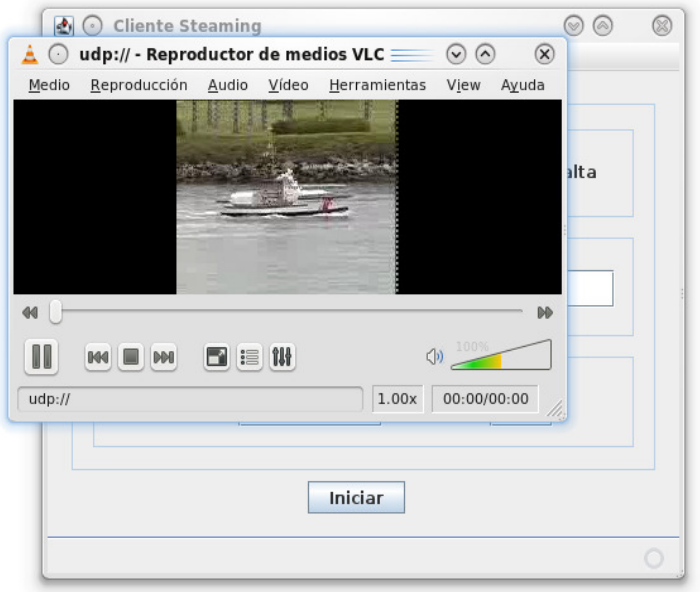


Figura 4.9 – Cliente reproduciendo vídeo.

Los Menús

En cuanto a los menús, no difieren de los explicados en el servidor, destacando únicamente el cuadro de diálogo “Acerca de…”.

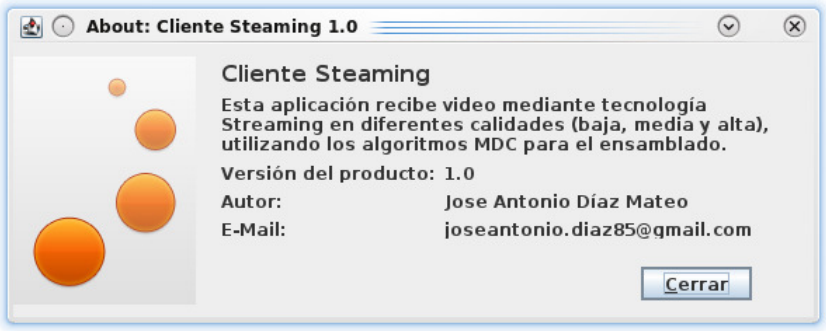


Figura 4.10 – Ventana de información del cliente.

Capítulo 5
Evaluación y prestaciones

5.1. Introducción

Este capítulo se dedica a estudiar los diferentes resultados que se han obtenido en una serie de pruebas que analizan parámetros como la calidad del video y el ancho de banda necesario para una correcta transmisión en función de dicha calidad para el algoritmo MDC en ambas formas. Esto nos permitirá decidir cuál es el más óptimo en cuanto a calidad y rendimiento.

5.2. La calidad del vídeo

A continuación, se muestran una serie de capturas para ambos algoritmos en las diferentes calidades para poder comparar cuál de las dos formas es más eficiente teniendo en cuenta cuánto degrada la imagen en las distintas calidades.

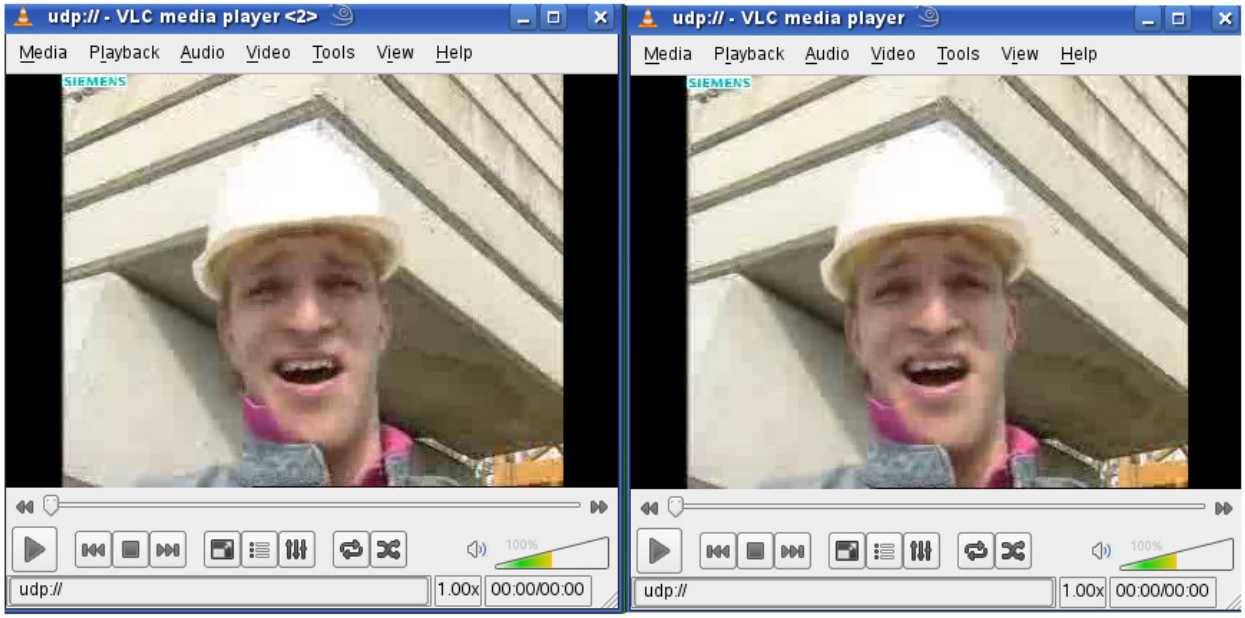


Figura 5.1 – Comparativa en calidad alta (MDC 1 a la izquierda y MDC 2 a la derecha).

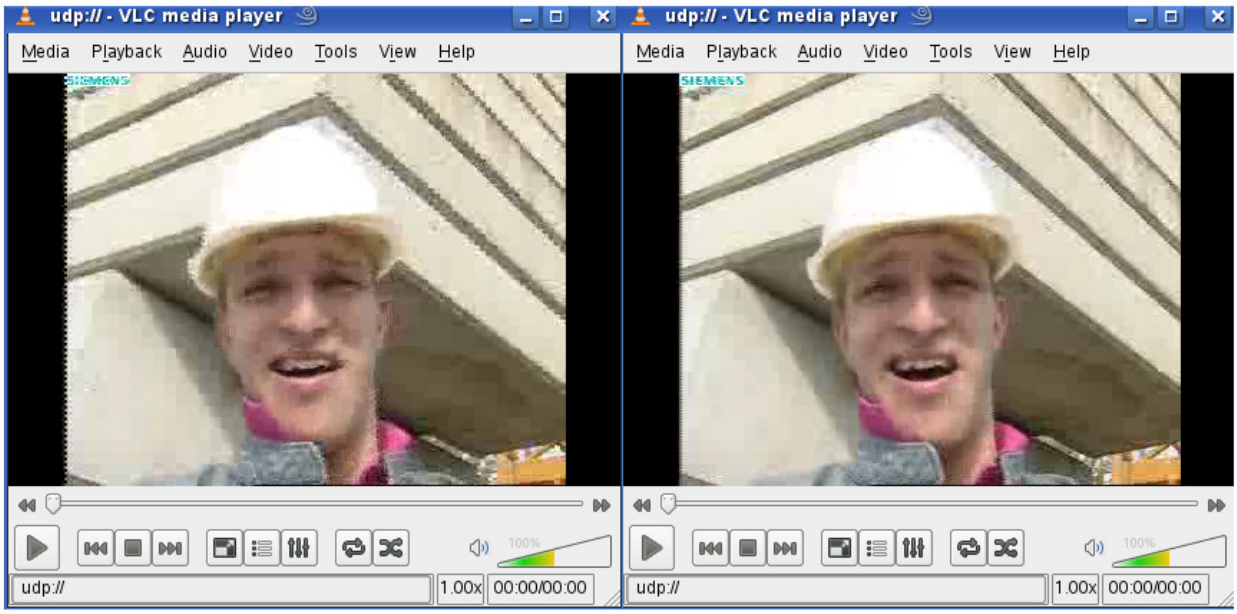


Figura 5.2 – Comparativa en calidad Media (MDC 1 a la izquierda y MDC 2 a la derecha).

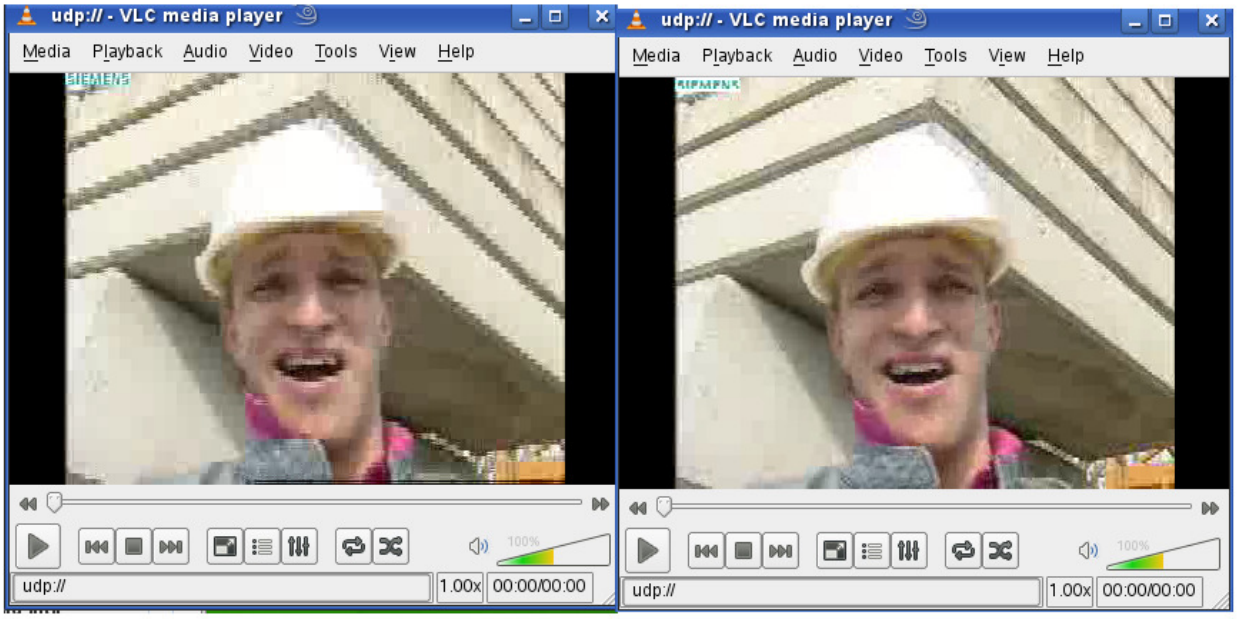


Figura 5.3 – Comparativa en calidad baja (MDC 1 a la izquierda y MDC 2 a la derecha).

Como se puede observar, El algoritmo MDC en su segunda forma, es el que ofrece mejores resultados, especialmente en la calidad más baja, donde la degradación de la imagen es máxima, es donde más se percibe la diferencia existente entre los dos. Esto se debe principalmente a la forma en que se estructuran los pixeles en el algoritmo divisor en las diferentes partes.

Capítulo 6
Conclusiones

6.1. Conclusiones

Hasta ahora se han planteado cuestiones como el Streaming, los algoritmos MDC o los formatos YUV. A modo de resumen global de todo lo que se ha tratado hasta ahora se pueden dividir las tareas que se han llevado a cabo en este proyecto en tres grandes grupos bien diferenciados.

Estudio sobre los formatos YUV y la Codificación Mediante Múltiples Descriptores (Multiple Description Coding o MDC). Aquí se han estudiado los diferentes formatos que nos encontramos dentro de la familia de los YUV o de Video Puro, y las diferentes formas en las que se puede tratar el video para degradar su calidad en función de una serie de criterios para el tratamiento de la sucesión de imágenes.

Estudio teórico para llevar a cabo el *Desarrollo de una Aplicación Cliente – Servidor* que utilice el Streaming y los MDC para intercambiar video en formato YUV, centrándonos en los del tipo 4:2:0. Se ha llevado a cabo un estudio teórico que nos ha permitido darle forma a lo que hoy es la aplicación, tomando como base los foramtos YUV y los algoritmos MDC.

Búsqueda de las Herramientas Adecuadas para llevar a cabo su implementación. Se han estudiado las diferentes posibilidades que nos ofrece el mundo de la programación y las herramientas externas, cuya combinación ha dado como resultado una aplicación funcional.

Bibliografía y Referencias

[1] YUV: Información sobre este formato y sus subformatos (En línea). URL:http://www.fourcc.org/yuv.php

[2] FFMPEG: Web oficial de la aplicación FFMPEG. Descarga de la aplicación, manual de instrucciones y soporte. (En línea). URL:http://ffmpeg.mplayerhq.hu

[3] JAVA Sun: Web oficial de Java Sun, (en línea). URL:http://java.sun.com