Transmisión Multicast de Vídeo MPEG-4

Ingeniería Técnica en Informática de Sistemas

José Miguel Dana Pérez
Director: Vicente González Ruiz

15 de julio de 2004

Agradecimientos

En la realización de este proyecto han colaborado muchas personas, ya sea de forma directa o indirecta. Sin su apoyo ésto no podría haber sido posible. Mis más sinceros agradecimientos a todos.

En especial:

- A mis padres, parte de culpa de que haya llegado hasta aquí les corresponde a ellos. Gracias.
- A familiares y amigos en general, por aguantarme día a día... incluso en los malos.
- A Vicente, el director de este proyecto, por su profesionalidad y por haber estado ahí en todo momento.

Gracias.

El conocimiento que no se puede transmitir
de una mente a otra no es conocimiento.

Índice general

Índice de Figuras

Índice de Tablas

Prólogo

¿Cuál es la finalidad de los sistemas de computación? ¿Qué pretendemos diseñando computadores cada vez más potentes?

El cometido fundamental de los sistemas de computación es el de conseguir realizar tareas complejas y repetitivas de la forma más eficiente posible; pero esto no es más que la definición clásica, hoy día sabemos que la misión de las computadoras va mucho más lejos.

Desde la creación de las llamadas redes de difusión y la tremenda popularidad de la que hoy goza Internet nos encontramos con una serie de necesidades, anteriormente inexistentes, que ahora se deben cubrir (información, comunicación, entretenimiento, etc.)

Aquí es donde podemos encontrar la unión entre las redes de comunicación y los servicios multimedia; emisión de radio por Internet, retransmisión de conciertos en directo, charlas virtuales, etc. Todos estos servicios tienen una mayor demanda cada día y es misión de los sistemas informáticos hacerlos posibles.

El rendimiento de los procesadores crece de forma exponencial desde hace varias décadas, lo que permite realizar operaciones cada vez más complejas. Hace 10 años habría sido impensable realizar todas las operaciones que lleva a cabo el estándar de vídeo MPEG-4 para decodificar una sóla imagen de una secuencia de vídeo en un tiempo prudencial, hoy día y con un equipo actual se realiza esta operación más de 30 veces por segundo.

Desgraciadamente las redes de comunicación no crecen al mismo ritmo, de ahí que las técnicas de compresión hayan avanzado tanto en los últimos años y se hayan convertido en una herramienta de uso diario; cuando la cantidad de datos a transmitir crece mucho más rápido que el ancho de banda de nuestro enlace nos vemos obligados a utilizar técnicas alternativas que nos permitan realizar la transmisión en el menor tiempo posible (igual que intentamos que el tiempo de procesamiento sea mínimo cuando realizamos algún tipo de cálculo sobre nuestro procesador). Aquí es donde entran en juego las técnicas de compresión y los métodos de comunicación. Un buen algoritmo de compresión y una técnica de transmisión eficiente nos pueden permitir optimizar esta tarea de forma más que significativa.

El método de compresión de vídeo MPEG-4 está incluido en el grupo de algoritmos

de compresión con pérdidas (*lossy*) de última generación. Permite altas tasas de compresión sin una pérdida excesiva de calidad, con lo que se convierte en un método más que interesante para la transmisión de vídeo en redes poco confiables, como podría ser Internet.

Por otro lado las comunicaciones Multicast permiten optimizar la transmisión del lado del servidor, de forma que el enlace no se vea saturado debido a una gran cantidad de receptores y la carga de trabajo sobre el primero sea mínima.

Este proyecto viene a ser un análisis de las cuestiones anteriores y un intento de implementar una solución eficiente al problema mencionado.

Capítulo 1

Transmisiones en Redes de Computadoras

1.1. Introducción

La transmisión de datos entre distintas computadoras es la base de cualquier red informática. No tiene sentido hablar de una red de computadoras en donde no se realizan transmisiones de datos de un equipo a otro de forma sistemática, ya que aquí encontramos la principal finalidad de las mismas: **compartir información**.

En los siguientes capítulos hablaremos en repetidas ocasiones de las llamadas *capas de red*[1], que no son más que secciones en las que descomponer un modelo de red básico. Utilizaremos una distinción basada en el modelo OSI (*Open Systems Interconnection*), mostrada en la Tabla 1.1

Tabla 1.1: Modelo de capas utilizado.

5	Capa de Aplicación
4	Capa de Transporte
3	Capa de Red
2	Capa de Enlace
1	Capa Física

En esta ocasión las capas que deberían centrar toda nuestra atención son las capas situadas por encima de la Capa de Enlace.

La Capa de Red será la encargada de dar el soporte necesario para que los paquetes puedan ser dirigidos desde el equipo origen hasta el destino, pasando por los routers que sean necesarios y de la forma más efectiva posible. La Capa de Transporte tiene la misión de transportar los datos de una máquina a otra mediante algún tipo de protocolo – de los que hablaremos más en profundidad posteriormente – haciendo posible la comunicación dentro de la red. Según el protocolo utilizado su cometido puede ser extendido y no sólo centrarse en esto último. Sin embargo, esta comunicación sería inservible si no realizásemos una *traducción* de los datos para conseguir una información válida para el usuario, de ésto se encarga la Capa de Aplicación y será en ese entorno precisamente donde desarrollaremos la parte software de nuestro proyecto.

1.2. La Capa de Transporte

La capa de transporte se puede considerar una de las capas más importantes del modelo de red. Su misión, como se ha dicho anteriormente, es la de trasladar los datos de un equipo emisor a un receptor independientemente de los enlaces físicos y elementos software existentes entre ambos.

En la capa de transporte de Internet, que es la que nos interesa, podemos encontrar fundamentalmente dos protocolos:

1. **UDP** (*User Datagram Protocol*), el cual permite el envío de datagramas encapsulados, básicamente el IP con una pequeña cabecera.

2. **TCP** (*Transmission Control Protocol*), que está especialmente diseñado para permitir un flujo de datos confiable dentro de una red poco confiable, como podría ser Internet.

1.2.1. El Protocolo UDP

El protocolo UDP consta básicamente una serie de cabeceras que se añaden a los datos a transmitir para permitir la distinción entre *computadora destino* y *proceso destino*. Así conseguimos que un mismo receptor pueda obtener flujos de datos distintos dirigidos a procesos distintos de forma concurrente. Evidentemente no debemos identificar los procesos directamente ya que éstos se crean y se destruyen constantemente; en su lugar el UDP hace uso de los llamados *puertos de protocolo* que se verán representados como un número entero positivo.

El UDP no asegura la llegada ni el orden de recepción de los mensajes; de hecho, cuando hablamos de transmisiones UDP hablamos de transmisiones unidireccionales, es

decir, a menos que el receptor comunique deliberadamente con el emisor no se transmitirá ninguna trama hacia el mismo, característica que, como veremos más adelante, no comparte el TCP, sobre el que hay una comunicación bidireccional con la finalidad de asegurar ciertas características inexistentes en el UDP.

Como podemos ver nos encontramos ante un protocolo muy simple, emplea el IP para llevar los mensajes a su destino y únicamente agrega una característica nueva a su funcionamiento: la capacidad de distinguir entre varios destinos dentro de un mismo receptor. Este capacidad, aunque simple, es fundamental en cualquier sistema de red que se precie.

El Formato de los Mensajes UDP

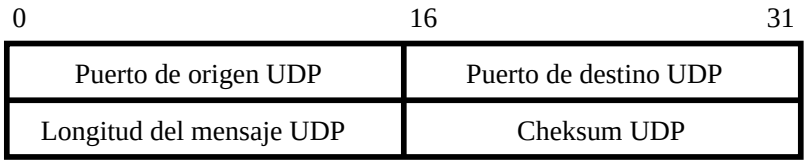


Figura 1.1: Cabecera de un datagrama UDP.

Los mensajes UDP son conocidos como *datagramas de usuario*; cada uno de los mismos consta de dos partes fundamentales: la cabecera UDP y el área de datos. Tal y como podemos apreciar en la Figura 1.1 el formato de la cabecera de los datagramas UDP es extremadamente simple, los puertos de origen y de destino permiten distinguir entre varios destinos dentro de un mismo receptor – tal y como apuntamos anteriormente – y, además, posee un campo dedicado al *Cheksum* del datagrama cuya misión es la de localizar posibles fallos en la transmisión. Éste, a pesar de ser opcional, es utilizado normalmente ya que es la única[1] forma de garantizar que el datagrama ha llegado a su destino tal y como salió del origen.

1.2.2. El Protocolo TCP

El TCP pretende ser un protocolo confiable ya que, en contraposición al UDP, éste asegura la llegada de los datos y mantiene el orden de los mismos. Ésto, junto a otras cuestiones que destacaremos a continuación, lo hace el protocolo de uso en Internet.

[1] El IP, protocolo de la capa de red de la que hablaremos más tarde, no implementa la suma de verificación para los datos.

Características de una conexión TCP

Al ser el protocolo TCP orientado a conexión, cuando una aplicación necesita realizar una comunicación ésta informa al sistema operativo. Ésto da lugar una serie de llamadas entre emisor y receptor mediante las cuales se acepta la conexión y se establece el instante en que se empezarán a transmitir los datos.

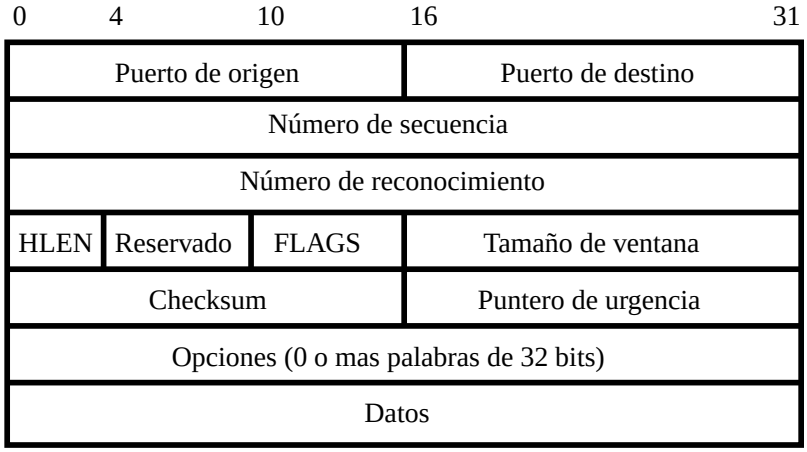


Figura 1.2: Segmento TCP.

La técnica utilizada por el TCP es la de *acuse de recibo positivo con retransmisión*; el emisor guarda un registro con los paquetes enviados y espera a que llegue el acuse de recibo (ACK) antes de enviar el siguiente. A su vez, implementa un temporizador que es lanzado en el momento del envío, de tal forma que si el ACK no llega antes de que éste expire se retransmitirá el segmento de datos. Esto último podría ocasionar una duplicación de las tramas recibidas, por eso se utilizan los llamados *números de secuencia*, que podemos ver en la Figura 1.2 correspondiente al formato de una segmento TCP.

Aún así seguimos teniendo un problema. El tener que esperar hasta la llegada del ACK correspondiente al segmento de datos enviado supone una latencia enorme en nuestra transmisión. Para solventarlo se utiliza la llamada *técnica de la ventana deslizante*, que consiste en colocar la ventana sobre la secuencia de datos y enviar los paquetes que residan dentro de la misma. Una vez recibido el ACK correspondiente al primero de ellos podemos mover la ventana y seguir con el paquete siguiente, y así sucesivamente hasta terminar nuestra comunicación.

Además las conexiones TCP conllevan una ventaja añadida sobre las transmisiones UDP y es que el TCP identifica las mismas por medio de un *par de puntos extremos*, con

lo que varias conexiones en la misma máquina pueden compartir el mismo número de puerto[2], lo que no ocurría con el UDP.

Mediante las técnicas citadas podemos asegurar que los datos llegan correctamente a su destino (acuses de recibo) y de forma ordenada (números de secuencia). Ésto supone el punto fuerte de este protocolo de transporte, porque cuando una aplicación trabaja sobre el TCP podemos estar seguros de que todos los paquetes llegarán a su destino, aunque para ello haga falta retransmitir gran parte de ellos. Sin embargo, para ciertos cometidos (como el relativo a este proyecto) esta característica se convierte en un impedimento. Si lo que pretendemos es realizar una transmisión en tiempo real de vídeo de la forma más eficiente posible el TCP no es nuestro protocolo por 3 motivos. Las retransmisiones anularían el primer concepto, (el de tiempo real) y los acuses de recibo provocarían una trabajo *extra* para el equipo emisor, así como un consumo de ancho de banda innecesario.

1.3. La Capa de Red

Esta capa es la encargada de guiar los datos desde el origen hasta su destino, realizando los *saltos* necesarios a través de los posibles routers de la forma más efectiva posible.

El protocolo más conocido de la Capa de Red es el **IP** (*Internet Protocol*), por ser el utilizado en Internet. Por ello todo lo que comentemos en los siguientes puntos estará relacionado con éste. Estudiaremos los tres tipos de comunicación básicos: Unicast, Broadcast y Multicast y evaluaremos las características de cada uno de ellos con la finalidad de seleccionar el que será más eficiente para nuestro trabajo.

1.3.1. Comunicación Unicast

Este tipo de comunicación es la más simple de todas. Un equipo emisor envía datos a un equipo receptor, por tanto es una comunicación directa entre los dos equipos.

El protocolo de transporte utilizado normalmente para este tipo de comunicaciones es el TCP. Ésto es bastante lógico dado que en una transmisión entre dos equipos nuestra intención será, en la mayoría de los casos, la de asegurar la recepción de los datos enviados (recordemos que UDP no era una protocolo confiable).

Si utilizásemos comunicaciones Unicast para transmisiones como la que deseamos realizar en este trabajo el resultado sería el mostrado en la Figura 1.3 para redes de área local (LAN) y por la Figura 1.4 para redes de área mundial (WAN), como podría ser Internet.

En el primer caso necesitaríamos crear un flujo de datos independiente para cada receptor, lo que provocaría un consumo excesivo del ancho de banda de la red que afectaría

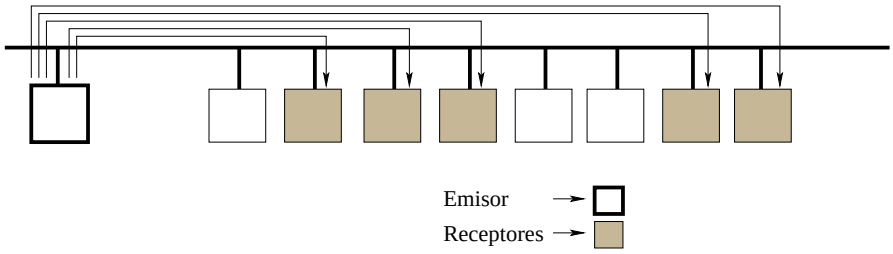


Figura 1.3: Transmisión Unicast en LAN.

a todos los equipos de la misma (incluidos los que no están recibiendo datos de parte del equipo emisor).

En el segundo caso también necesitaríamos un flujo de datos por receptor, pero la sobrecarga no afectaría a todos los equipos. No obstante, los enlaces cercanos al emisor sí se verían congestionados.

Pero no sólo debemos centrarnos en el problema de la saturación del enlace, también encontraríamos una sobrecarga desde el punto de vista computacional por parte del emisor, ya que debería enviar una copia distinta de datos para cada receptor. Sí además utilizásemos TCP tendríamos que gestionar las distintas conexiones de forma independientemente, añadiendo el control sobre los ACK y los reenvíos, además de imposibilitar una transmisión en tiempo real como ya comentamos anteriormente.

Por tanto, podemos concluir que una comunicación Unicast no es precisamente la forma óptima de llevar a cabo una transmisión como la que pretendemos.

1.3.2. Comunicación Broadcast

Las comunicaciones Broadcast permiten llevar un flujo de datos único a todos los equipos de la red. Estamos por tanto ante una forma de solucionar el problema de tener que mantener un flujo de datos independiente para cada receptor (Unicast).

Sin embargo no estamos ante una solución óptima, ya que habría equipos que no deseen recibir los datos enviados y, sin embargo, éstos llegaran de igual forma que a los que sí lo desean. Además nos encontraríamos ante una situación en donde la seguridad se vería comprometida seriamente ya que no podríamos controlar los equipos que acceden a los datos enviados. Por consiguiente, la única condición para acceder a ellos sería pertenecer a la LAN.

En la Figura 1.5 podemos ver un ejemplo de comunicación Broadcast y como todos los equipos reciben los datos a pesar de no haberlo solicitado.

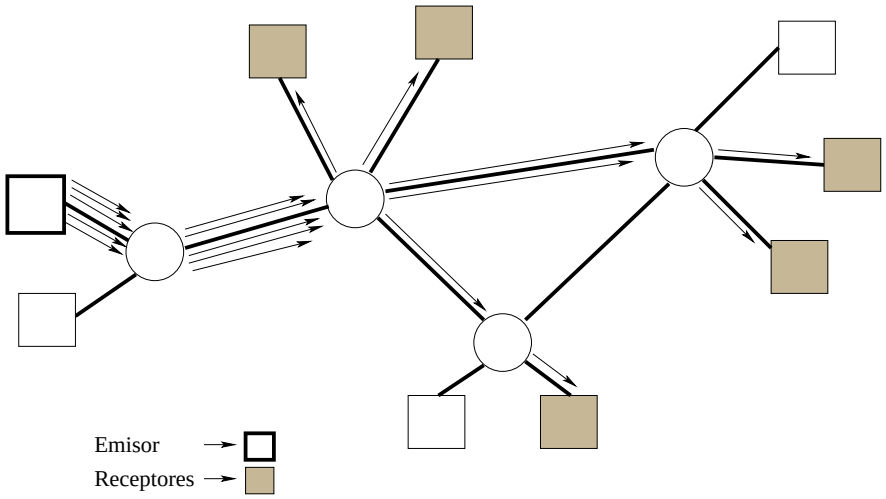


Figura 1.4: Transmisión Unicast en WAN.

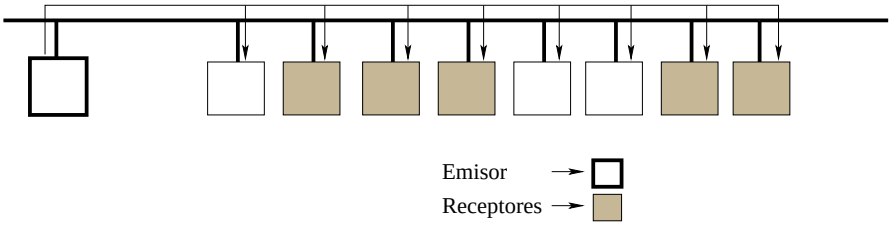


Figura 1.5: Transmisión Broadcast en LAN.

Además, la comunicación Broadcast se ve limitada al ámbito local. Serían inimaginables las consecuencias causadas por una transmisión Broadcast a nivel mundial. Si alguien tuviese la posibilidad de realizar envíos a todos los equipos de, por ejemplo, una red como Internet la saturación generada dentro de la misma haría la comunicación prácticamente imposible.

1.3.3. Comunicación Multicast

Llegamos hasta el tipo de comunicación elegida para el desarrollo de nuestro proyecto. La primera implementación de Multicast se realizó en 1993, en la versión 4.4 de BSD [3], sin embargo nos encontramos ante un tipo de comunicación que no se encuentra

suficientemente explotada; ésto es debido, en parte, a las carencias que tiene la versión 4 del protocolo IP (en adelante IPv4) con respecto a este tipo de comunicaciones. El funcionamiento de una comunicación Multicast es bastante simple y la describiremos a continuación.

Existe un *canal Multicast* al que distintos receptores pueden suscribirse; por otra parte, un equipo emisor se encarga de transmitir los datos hacia el canal, de forma que exista **un único flujo de datos**. Los receptores son los encargados de conectar con el canal para recoger los datos que envía el emisor al mismo. Así éste se ve liberado de tener que controlar los envíos hacia los receptores como sucedía con el Unicast. Además los receptores son únicamente aquellos que realmente quieran acceder a los datos, al contrario de lo que sucedía con el Broadcast en el que todos los equipos de la red eran receptores. Por supuesto una comunicación de este tipo es imposible de realizar con un protocolo de transporte confiable como TCP, ya que el emisor no puede ni debe gestionar conexiones hacia los receptores. Es más, no tiene ni por qué saber el número de receptores que están accediendo al canal.

Direcciones Multicast

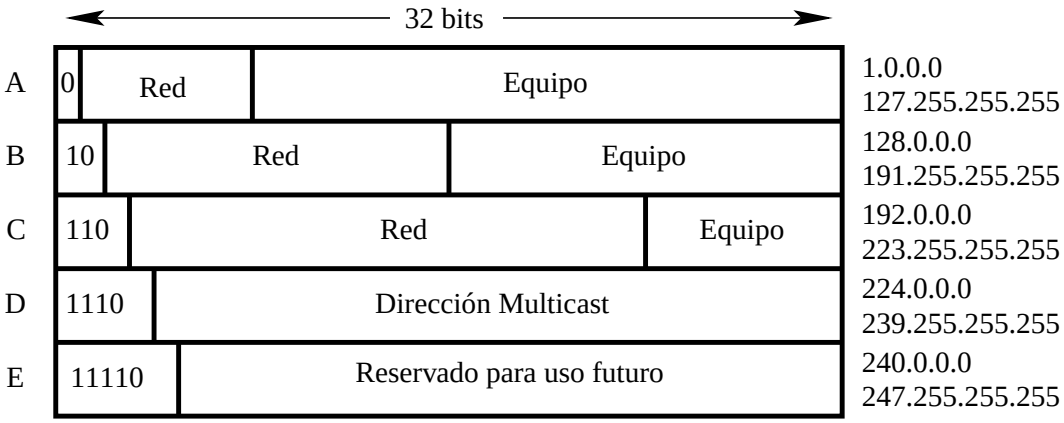


Figura 1.6: Rangos de direcciones IP.

Las direcciones IP Multicast son direcciones de clase D (ver Figura 1.6) y dentro de este rango el comprendido entre 224.0.0.0 y 224.0.0.225 se encuentra reservado; 224.0.0.1, por ejemplo, representa a todos los equipos de la LAN, 224.0.0.2 a todos los routers Multicast de la LAN, 224.0.0.4 todos los routers DVMRP (*Distance Vector Multicast Routing*), etc.

En la Figura 1.6 vemos que disponemos de 28 bits para identificar la dirección o grupo Multicast, con lo que más de 260 millones de grupos pueden coexistir sin problemas en Internet y estar transmitiendo simultáneamente.

El Multicast Backbone

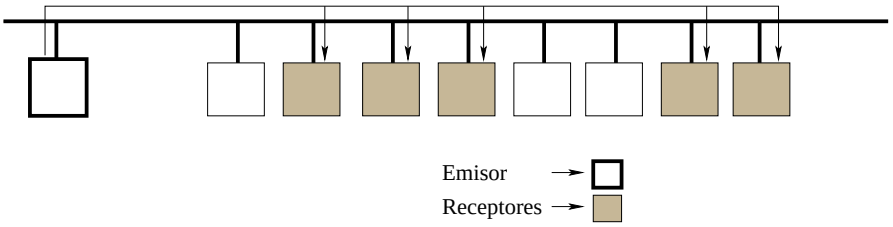


Figura 1.7: Transmisión Multicast en LAN.

En la Figura 1.7 vemos un ejemplo de transmisión Multicast en una LAN. Dentro de la misma el envío de los datagramas UDP se realiza sin problema alguno, al contrario que en redes de ámbito mundial donde, como veremos a continuación, necesitaremos algo más de *ayuda* para que los receptores puedan suscribirse a nuestro canal.

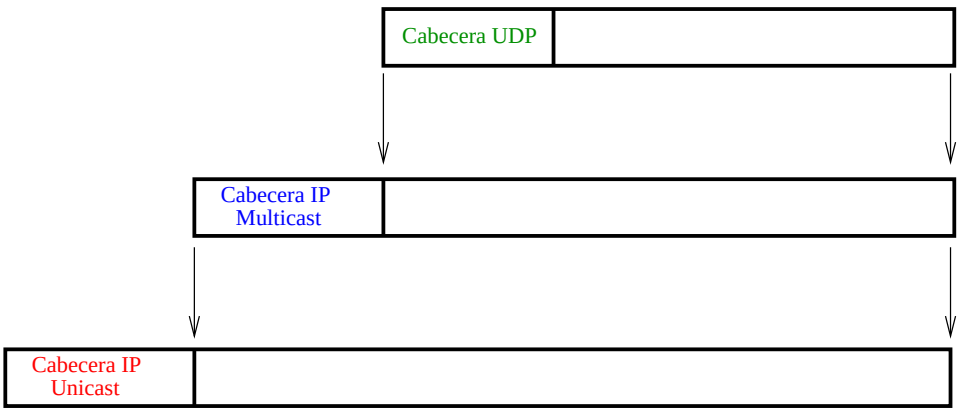


Figura 1.8: Funcionamiento del IP-in-IP.

Como decíamos, a pesar de todas las ventajas que nos puede aportar una comunicación de este tipo ésta no se utiliza comúnmente para las transmisiones, por ejemplo, de

conciertos, charlas, etc. a través de Internet. Ésto es debido a que el *enrutado* de los paquetes IP requiere de *routers Multicast* que realicen la operación que podemos ver en la Figura 1.8.

Al igual que los paquetes UDP son encapsulados dentro de paquetes IP, los paquetes IP Multicast son encapsulados dentro de paquetes IP Unicast, implementando lo que llamamos IP-in-IP. Ésta operación no es sólo realizable por routers Multicast, también podemos realizar el encapsulado por software.

Éste es precisamente el modo de funcionamiento del MBONE (*Multicast Backbone*). El MBONE está basado en el *tunneling*, que consiste en realizar túneles entre los distintos routers Multicast que existen en Internet con la finalidad de poder hacer llegar las comunicaciones a todos ellos, mediante el método de IP-in-IP descrito anteriormente. De esta forma se crean las llamadas *islas Multicast*, que no son más que redes de área local conectadas a Internet mediante un router Multicast que les informa de los grupos que existen actualmente dentro del MBONE y a los que puede suscribirse.

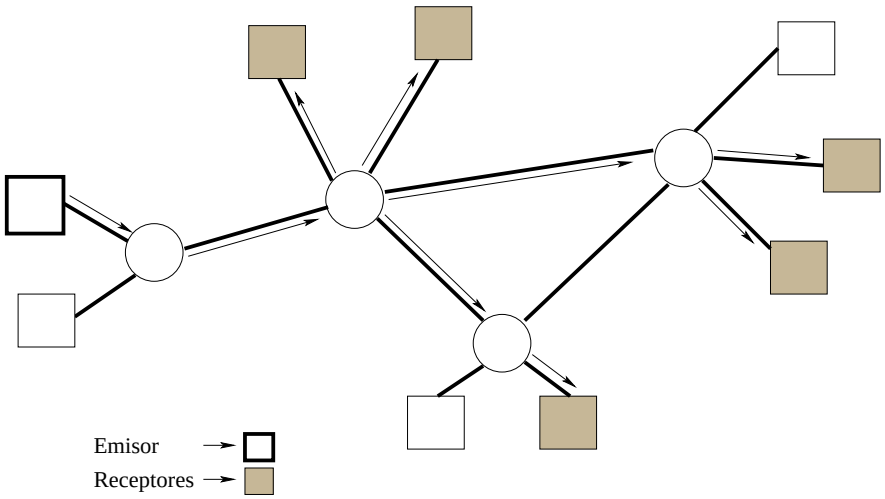


Figura 1.9: Transmisión Multicast en WAN.

El funcionamiento de una transmisión Multicast a nivel de WAN es el mostrado en la Figura 1.9 (suponemos que los routers son routers Multicast).

Como podemos ver el emisor no tiene que controlar varias comunicaciones para enviar los datos a tantos receptores como lo deseen. Ésto se traduce en un ahorro en el ancho de banda, sobre todo en los tramos de red cercanos al mismo. Sobre cómo los routers encaminan los datos únicamente hacia las LAN en las que existen receptores hablaremos en el siguiente apartado, dedicado al protocolo IGMP.

El IGMP (*Internet Group Management Protocol*) [4]

Como hemos dicho los routers Multicast son los encargados de, mediante el IP-in-IP, implementar el *tunneling* que permita la comunicación del MBONE. Sin embargo, esta operación no es la única que realizan los encaminadores. Además cada uno de ellos deberá informar a los equipos pertenecientes a su LAN de la existencia de los distintos grupos Multicast que se encuentren emitiendo en ese momento dentro del MBONE.

Existen varias versiones del IGMP, de entre las que comentaremos las siguientes:

1. Versión 1: Es el estándar original del IGMP y se encuentra descrito en el RFC-1112.
2. Versión 2: Es el más utilizado en Internet, a pesar de no ser y -supuestamente- que nunca llegará a ser un estándar, dado que la versión 3 lo ha dejado obsoleto. Se encuentra descrito en el RFC-2236.
3. Versión 3: Propuesta de estándar realizada en Octubre del 2002, descrita en el RFC-3376 y soportado por los sistemas operativos actuales.

IGMPv1

En esta versión del IGMP tenemos dos tipos de mensajes:

1. Mensajes de consulta (realizados por los routers).
2. Mensajes de informe (realizados por los equipos).

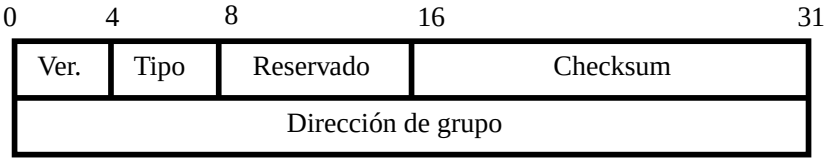


Figura 1.10: Mensaje IGMPv1.

Y la estructura básica es la que podemos ver en la Figura 1.10, donde `Ver.` indica la versión del protocolo (en este caso 1), `Tipo` indica si es un mensaje de consulta (1) o de informe (2), el `Checksum` se realiza de igual forma que en el protocolo IP y la `Dirección de grupo` está a 0 cuando el mensaje es de consulta.

El funcionamiento básico de este protocolo es muy simple:

1. En cada enlace de la red uno de los routers (si es que existen varios) será el encargado de realizar las consultas (aproximadamente una cada 60-90 segundos). Éstas se realizarán mediante un mensaje al grupo reservado a todos los equipos de la LAN (224.0.0.1) con un TTL=1.

2. Los receptores del mensaje (todos aquellos que soporten recepción Multicast) iniciarán una serie de contadores aleatorios (entre 0 y 10) para cada uno de los grupos a los que pertenezcan.

3. Cuando el contador de un equipo (`A`) para un determinado grupo (`G`) expira éste manda un *mensaje de informe* a `G` con TTL=1.

4. Los demás miembros de `G` *escuchan* el informe de `A`, paran sus contadores y cancelan cualquier informe pendiente.

5. El router encargado de la encuesta seguirá escuchando todos los informes e ignorará los mensajes de los grupos no solicitados.

Como vemos, en esta versión del protocolo (y en la siguiente) no es necesario que informen todos los equipos de un mismo grupo. A los routers sólo les hace falta conocer si hay algún interesado en que los datagramas lleguen a la LAN encapsulados en segmentos IP Multicast. De esta forma los routers no pueden ni deben saber el número de clientes interesados en recibir datos de un grupo determinado.

Sin embargo, cuando un equipo quiere iniciar una comunicación con un grupo al que no pertenece envía uno o dos informes sin esperar una consulta por parte del router. Ésto último es bastante lógico dado que, de ser el único equipo interesado en ese grupo concreto, no tendría forma de conectarse al mismo por el método citado anteriormente, ya que al no pertenecer al mismo no se iniciaría ningún contador y no podría enviar el informe tras una consulta.

IGMPv2

El IGMPv2 nace como una versión que pretende reducir las latencias ocasionadas por el funcionamiento de su antecesora de forma que, partiendo del método básico de encuesta, se implementan nuevas características, manteniendo la compatibilidad hacia atrás.

El formato del mensaje (Figura 1.11) es prácticamente idéntico al de la versión anterior, salvo que los campos `Versión` y `Tipo` se unifican en este último, de forma que podremos encontrar los siguientes valores en el mismo:

- `0x11` - Mensaje de consulta.

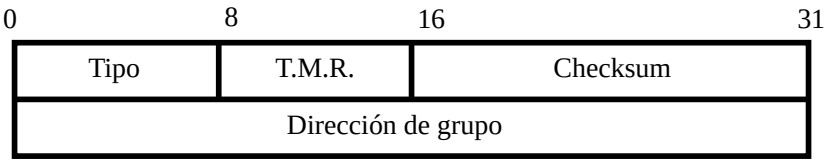


Figura 1.11: Mensaje IGMPv2.

- `0x12` - Mensaje de informe (versión 1).
- `0x16` - Mensaje de informe (versión 2).
- `0x17` - Abandonar el grupo.

El `T.M.R.` (*Tiempo Máximo de Respuesta*) indica el tiempo (expresado en décimas de segundo) que esperará el router tras una consulta antes de ignorar el grupo correspondiente. En este caso `Dirección de grupo` será 0 sólo cuando se encuesten todos los grupos, ya que con esta nueva versión se podrán hacer encuestas sobre grupos individuales.

La técnica fundamental para disminuir los tiempos de latencia ocasionados por la versión anterior será la siguiente:

1. Cuando en equipo abandone un grupo enviará el mensaje correspondiente (`0x17`) *si ha sido el equipo más reciente en enviar un mensaje de informe sobre ese grupo* (lo que indicaría que puede ser el único perteneciente al mismo).

2. Cuando el router recibe el mensaje de abandono de grupo envía una consulta específica para ese grupo concreto, especificando el tiempo máximo de respuesta establecido para los mensajes de informe.

3. En caso de no recibir ningún mensaje de informe en el tiempo máximo establecido se entiende que no queda ningún equipo suscrito a dicho grupo y, por consiguiente, deja de enviar segmentos IP Multicast a esa LAN.

Como hemos visto, con esta nueva versión del protocolo podemos reducir el ancho de banda inútil del enlace que suponen los datos de grupos a los que no hay ningún equipo suscrito de forma inmediata.

IGMPv3

Esta *nueva* versión permite a los routers conocer cuántos equipos escuchan a un determinado grupo. Ésto significa que cada uno de los equipos suscritos deberá enviar un mensaje de informe para poder continuar recibiendo datos (y no sólo uno como en versiones anteriores).

Se mantiene la compatibilidad hacia atrás, aunque el método de funcionamiento es distinto ya que, por ejemplo, los mensajes de informe son enviados a 224.0.0.22 (dirección Multicast reservada para permitir la comunicación con los *switches* de la capa de enlace).

Además implementa funciones relacionadas con la seguridad, como puede ser la protección ante ataques de denegación de servicio (*DoS*). Aún así sigue siendo un *borrador* y es difícil encontrarlo implementado fuera del ámbito de la investigación.

IGMP Snooping

En redes locales podemos encontrar una serie de elementos intermedios que hacen disminuir la efectividad de las comunicaciones Multicast.

Podemos calificar como elementos intermedios a aquellos componentes hardware que encontramos entre los routers y los hosts. Ejemplos de éstos pueden ser switches y hubs. Según su funcionamiento *clásico* éstos envían el tráfico Multicast a todos los nodos conectados, aunque no haya ningún receptor suscrito al grupo en cuestión. Ésto provoca un gasto del ancho de banda en las partes del enlace donde el tráfico Multicast no es necesario, aquí es donde el *IGMP Snooping* nos permite optimizar la transmisión.

Los switches que implementan el IGMP Snooping miran dentro de las tramas Multicast con la finalidad de conocer las direcciones de los receptores suscritos (por medio de los mensajes de informe IGMP) y las direcciones de los routers Multicast (por medio de los mensajes de consulta IGMP). De esta forma los paquetes Multicast son sólo transmitidos por los enlaces necesarios de la red, sin consumir ancho de banda innecesario.

El funcionamiento completo se puede ver en la Figura 1.12. Por supuesto es extensible a varios grupos, que podrían convivir en la misma red de forma independiente y sin que sus paquetes se direccionen hacia objetivos no necesarios.

Pero el IGMP Snooping no está carente de problemas, los cuales pasamos a enumerar a continuación:

1. El que los switches tengan que analizar los paquetes Multicast para encontrar la dirección del remitente supone un coste que incrementa la latencia. Afortunadamente ésto se encuentra parcialmente solucinado por el IGMPv3, ya que los mensajes de informe se envían a un grupo fijo (224.0.0.22), con lo que los switches sólo tienen que encuestar a éste.

2. No hay una compatibilidad entre versiones. Los switches preparados para el IGMPv2 ignorarán los mensajes proporcionados por el IGMPv3.

3. Las reglas de Snooping son complejas y no hay un estándar. Cada fabricante tiene las suyas propias, con lo que también encontramos incompatibilidades[2].

Routing Multicast

Hay distintos protocolos para realizar routing en comunicaciones Multicast. Los más conocidos y utilizados son los siguientes:

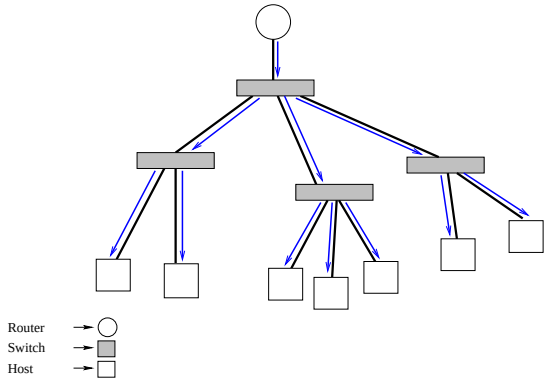
1. **DVMRP *Distance Vector Multicast Routing Protocol***: Basado en la *poda* de los árboles generados por los grupos Multicast. Construye su propia tabla de routing.

2. **MOSPF *Multicast Extensions to OSPF***: Basado en la consulta de miembros por broadcast. Utiliza la tabla de routing del OSPF.

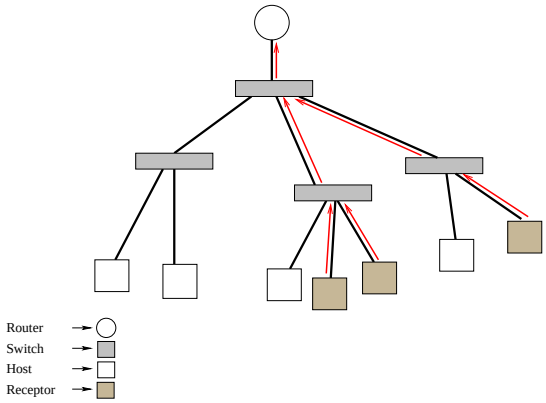
3. **PIM-DM *Protocol Independent Multicast, Dense-Mode***: Parecido al DVMRP, en cuanto a que utiliza el mismo sistema de poda. Sin embargo, la tabla de routing es una tabla de routing Unicast.

4. **PIM-SM *Protocol Independent Multicast, Sparse-Mode***: Cuando los receptores envían mensajes de informe a los routers éstos transmiten a su vez un mensaje RP (*Rendezvous Point*) al router más próxmo al emisor Multicast. Acto seguido pasa a formar parte del árbol compartido y se inicia una secuencia de *poda* y de búsqueda del camino más corto con la finalidad de optimizar el arbol compartido.

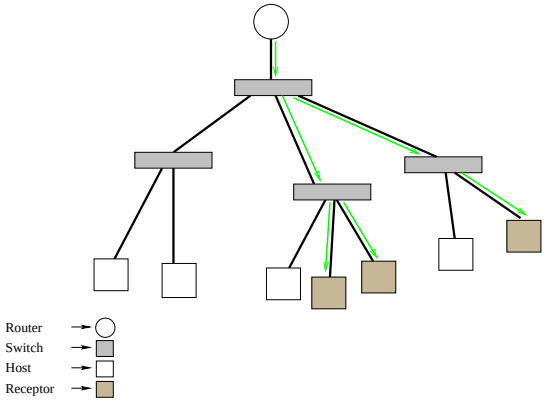
[2]Una prueba de ésto es el CGMP (*Cisco Group Management Protocol*), que es incompatible con el IGMP Snooping (en los modelos de Cisco de menor calidad).



(a) Mensaje de consulta IGMP.



(b) Mensajes de informe IGMP.



(c) Los paquetes sólo se envían a las partes del enlace necesarias.

Figura 1.12: Funcionamiento del IGMP Snooping.

Capítulo 2

El Sistema de Codificación de Vídeo MPEG-4

2.1. Introducción

El formato de vídeo MPEG-4, desarrollado por el MPEG (*Moving Picture Experts Group*) y finalizado en Octubre de 1998 se convierte en estándar internacional (ISO/IEC 14496) en los primeros meses de 1999. A finales de 1999 este mismo grupo crea la extensión conocida como *MPEG-4 Versión 2*, la cual adquiere el reconocimiento de estándar a principios del año 2000.

Con la revolución de la televisión digital, las aplicaciones gráficas interactivas, las transmisiones *en vivo* de grandes acontecimientos por Internet y, en general, de los sistemas multimedia y las redes globales de comunicación, el sistema de codificación MPEG-4 viene a ser una *necesidad* más que un avance en el mundo del tratamiento de vídeo.

Este formato nos permite obtener tasas de compresión suficientemente elevadas sin una pérdida excesiva de la calidad final. Este hecho, unido a la posible escalabilidad, lo convierte en un sistema ideal para las transmisiones de vídeo en redes donde el ancho de banda es limitado.

Además, tal y como explicaremos en los siguientes apartados, MPEG-4 nos permite dividir las escenas en *objetos*, de forma que podamos tratar de distinta forma secciones de una secuencia (fondos estáticos, caras, voces, etc.). Así mismo también será posible *insertar* elementos sintéticos dentro de una escena real, lo que nos permitirá personalizar en tiempo real las imágenes resultantes.

Concretamente estudiaremos el algoritmo de codificación para escenas rectangulares, ya que es el que se encuentra implementado de forma generalizada actualmente y es, podríamos decir, el único que se utiliza de forma masiva.

El MPEG-4 como sistema de codificación abarca un amplio abanico de posibilidades

como reconocimiento facial, de voces, etc. Pero la mayoría de estas características o no están implementadas o se encuentran en investigación actualmente.

2.2. Los Elementos de una Secuencia de Datos

Una de las principales diferencias entre el MPEG-4 y, por ejemplo, sus antecesores MPEG-1 y MPEG-2 es que éste permite representar las escenas como un conjunto de objetos audiovisuales. De ésta forma se pierde *levemente* el concepto de *frame* (una imagen dentro de una secuencia).

Además tenemos la posibilidad de representar el audio, vídeo, texto, etc. como objetos independientes, tratándolos de forma distinta y, de esta forma, aplicar distintas técnicas de compresión a cada uno de ellos con la finalidad de obtener unos mejores resultados. Finalmente unimos y sincronizamos todos los objetos independientes para formar las escenas finales, pudiendo incluso *insertar* componentes sintéticos para alterar las imágenes en tiempo real [5]. Ésto último puede ser de utilidad para la publicidad de eventos retransmitidos a distintos países, ya que podríamos sustituir los carteles de los anunciantes según el país receptor mediante imágenes artificiales con la finalidad de personalizar la publicidad y obtener unos mayores beneficios.

La compresión de vídeo MPEG-4 explota las redundancias tanto espaciales como temporales que ocurren en las secuencias de vídeo. La redundancia espacial se elimina codificando cada frame por separado, como si de un algoritmo de compresión de imágenes se tratase; ésta técnica se conoce como codificación *intraframe*. Sin embargo la codificación que nos permitirá obtener las mayores tasas de compresión será la que se encargue de eliminar la redundancia temporal, producida por la gran similitud existente entre frames consecutivos; estamos hablando de las codificación *interframe*.

La codificación interframe se encargará de codificar las diferencias entre frames consecutivos, como si de un cuantificador incremental se tratase. Sin embargo, para escenas donde se produzcan grandes cambios, como puede ser un zoom o similares, esta técnica no funciona correctamente, por lo que MPEG-4 incorpora una *técnica de compensación de movimiento* que soluciona el problema.

En el MPEG-4 un objeto de vídeo puede ser un frame completo o sólo una parte del mismo (por eso decíamos anteriormente que se *pierde* el concepto de frame). Tenemos 3 tipos principales de *planos de objeto de vídeo* (en adelante VOP) [6]:

1. **I-VOPs**: Contienen imágenes con codificación intraframe, es decir, *frames* completos a los que no se les ha aplicado una codificación incremental.

2. **P-VOPs**: Se ha realizado una codificación predictiva sobre los mismos basándose en los VOPs previamente codificados.

3. **B-VOPs**: Se codifican de forma *bidireccional*, basándose en las diferencias con los VOPs tanto anteriores como posteriores.

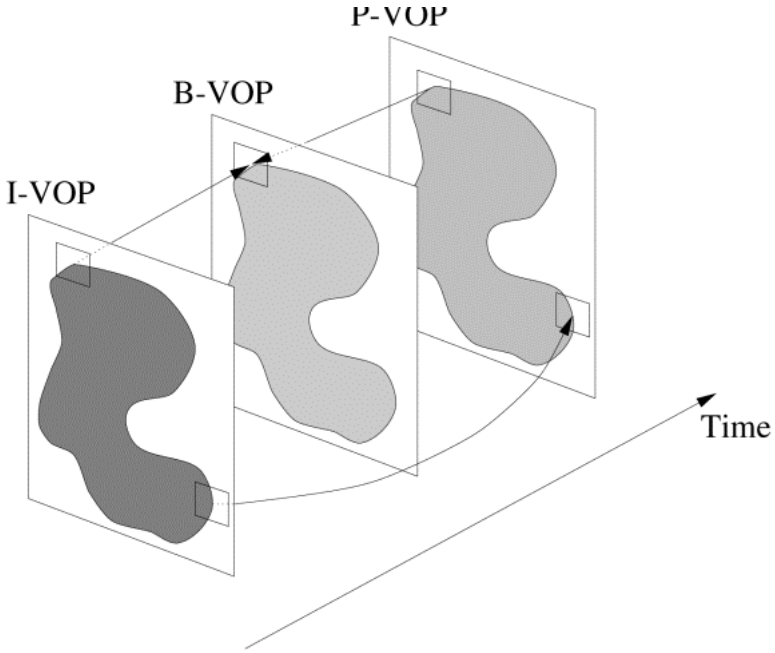


Figura 2.1: Relación entre VOPs.

Como podemos ver en la Figura 2.1, la relación existente entre los tres tipos de VOPs es totalmente dependiente, lo que provocará unas grandes tasas de compresión y más de alguna fuente de error, tal y como comentaremos más adelante.

Los I-VOPs deberán aparecer regularmente en la secuencia de datos ya que son necesarios para la descodificación de los P-VOPs y los B-VOPs.

Pero los VOPs son sólo los componentes más básicos de una secuencia de vídeo MPEG-4. En la Figura 2.2 podemos ver la estructura lógica general de un flujo de datos completo, el cual se estructura según los siguientes componentes [7]:

1. **Visual Object Sequence (VS)**: La escena MPEG-4 al completo, que puede contener objetos 2-D y 3-D, así como elementos naturales y sintéticos.

2. **Video Object (VO)**: Corresponde a un elemento en 2-D de la imagen, en el caso más simple puede ser un frame rectangular o un objeto formado arbitrariamente como el fondo de la secuencia.

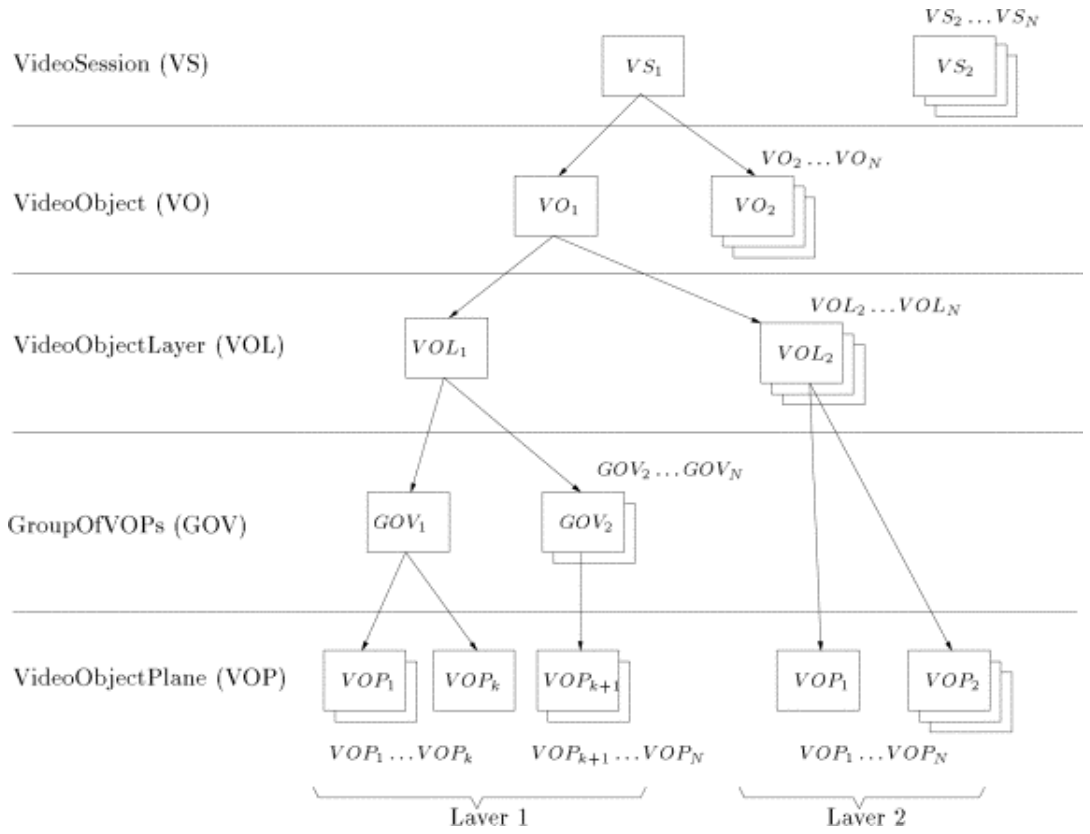


Figura 2.2: Estructura lógica de una secuencia de datos MPEG-4.

3. **Video Object Layer (VOL)**: Cada objeto puede ser codificado como escalable o no escalable, dependiendo de la aplicación. El VOL es el encargado de proporcionar soporte para codificación escalable, dependiendo del ancho de banda disponible, potencia de cálculo, etc. los parámetros pueden ser ajustados y proporcionados *a posteriori* en el descodificador. Dentro de esta capa podemos encontrar:

   a) **Group of Video Object Planes (GOV)**: Nos permiten agrupar VOPs para tratarlos conjuntamente, este elemento es opcional.

   b) **Video Object Plane (VOP)**: De los que ya hemos hablado anteriormente. Forman la capa más baja de elementos dentro de la estructura y son los que llevan realmente la información sobre la que trabajaremos.

Todos los VOPs consisten en *macrobloques*. Cada macrobloque es una matriz de

16x16 pixels si nos encontramos en el espacio de la luminancia o de 8x8 pixels si en el espacio de la crominancia para el muestreo de color más simple.

2.3. El Algoritmo de Codificación

Una vez conocidos los distintos objetos que forman parte de una secuencia de datos entraremos a estudiar el algoritmo de codificación del MPEG-4.

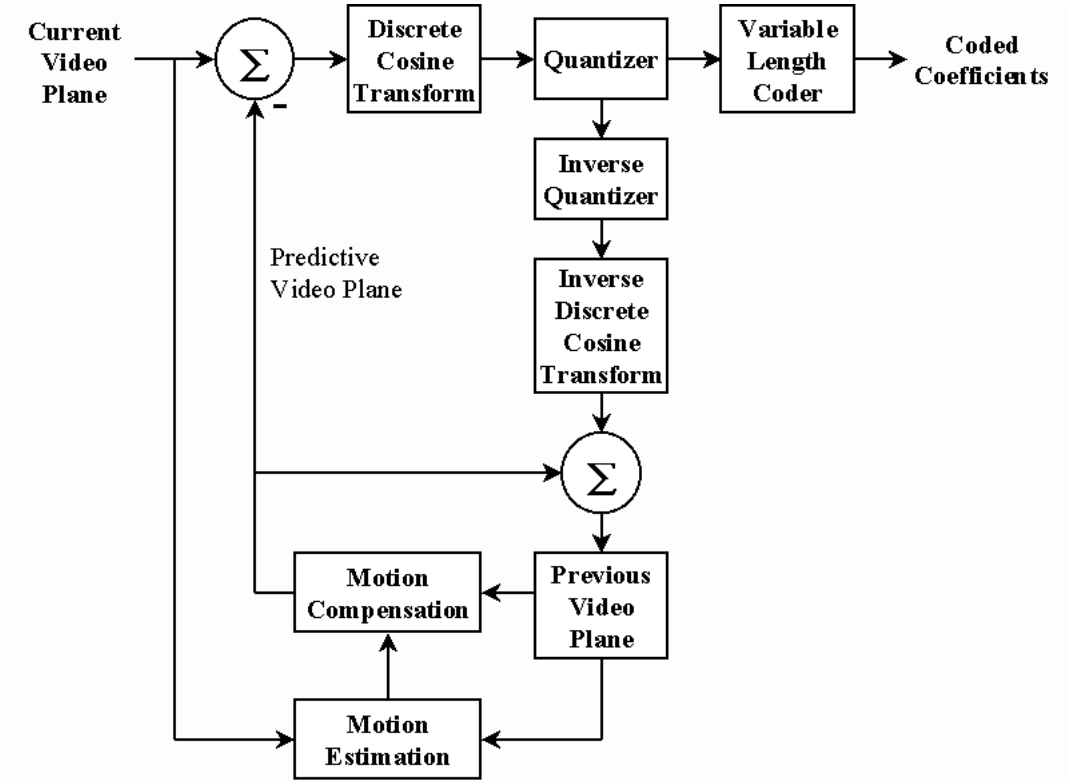


Figura 2.3: Diagrama de bloques del proceso de codificación.

El esquema básico de funcionamiento es el que podemos ver en la Figura 2.3.

Primero el codificador determina el tipo del VOP actual. Si éste es un I-VOP se le aplica la transformada discreta del coseno (ver Apéndice A) bi-dimensional a cada uno de los macrobloques en los que esté compuesto para pasar la información en el dominio espacial al dominio de la frecuencia.

2.3.1. Cuantificación de los Coeficientes de la DCT [8]

Acto seguido los coeficientes de la DCT se cuantifican. Ya que la mayor parte de la energía se acumula en los coeficientes de menor frecuencia espectral, podemos eliminar o utilizar menos niveles de cuantificación para los coeficientes de mayor orden. Ésto no supone una pérdida excesiva de la calidad final, dado que las altas frecuencias son filtradas de forma natural por el sistema visual humano.

El estándar MPEG-4 reconoce dos posibles algoritmos de cuantificación. El primero se deriva del estándar MPEG-2 y el segundo es usado en la recomendación ITU-T H.263.

El proceso de cuantificación está controlado por el parámetro `quantiser_scale`, que puede tomar valores desde 1 hasta $(2^{\texttt{quant\_precision}} - 1)$, donde `quant_precision` se puede inicializar desde 3 hasta 9, ambos inclusive. Además es posible modificar el valor de `quantiser_scale` según modos de macrobloques especiales.

Cuantificación de Coeficientes Intra-DC

Los coeficientes DC de los macrobloques intracodificados (a los que se les ha aplicado la codificación intraframe) son cuantificados utilizando un cuantificador no lineal optimizado.

Tabla 2.1: Relación entre el `dc_scaler` y el `quantiser_scale`.

`quantiser_scale` (Q_p)	1-4	5-8	9-24	25-31
`dc_scaler` (Luminancia)	8	$2Q_p$	$Q_p + 8$	$2Q_p - 16$
`dc_scaler` (Crominancia)	8	$(Q_p + 13)/2$	$(Q_p + 13)/2$	$Q_p - 6$

Contamos con un nuevo parámetro de especial importancia, el `dc_scaler`, que dependerá del valor de `quantiser_scale` según la Tabla 2.1.

De esta forma y una vez obtenido el valor del `dc_scaler` la cuantificación y su inversa son realizadas según las formas expresadas por las Ecuaciones (2.1) y (2.2), respectivamente.

$$QF[0][0] = F[0][0] // \texttt{dc\_scaler} \tag{2.1}$$

$$F[0][0] = QF[0][0] \cdot \texttt{dc\_scaler} \tag{2.2}$$

Donde la operación $//$ indica una división entera con redondeo al más cercano.

Primer Método de Cuantificación: Cuantificación MPEG

A pesar de ser un método derivado del estándar MPEG-2 se conoce comúnmente como *cuantificación MPEG* a secas.

Éste método es muy parecido al del estándar de imagen JPEG, ya que se adapta a las propiedades del *sistema visual humano* (SVH). Para ello emplea las llamadas *tablas de cuantificación*, de forma que cada coeficiente recibe un tratamiento distinto según el índice espectral que lo representa.

Tabla 2.2: Tabla de cuantificación para macrobloques intracodificados.

8	17	18	19	21	23	25	27
17	18	19	21	23	25	27	28
20	21	22	23	24	26	28	30
21	22	23	24	26	28	30	32
22	23	24	26	28	30	32	35
23	24	26	28	30	32	35	38
25	26	28	30	32	35	38	41
27	28	30	32	35	38	41	45

Tabla 2.3: Tabla de cuantificación para macrobloques intercodificados.

16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24
18	19	20	21	22	23	24	25
19	20	21	22	23	24	26	27
20	21	22	23	25	26	27	28
21	22	23	24	26	27	28	30
22	23	24	26	27	28	30	31
23	24	25	27	28	30	31	33

Se utilizan tablas distintas según el tipo de codificación aplicada a los macrobloques, de forma que la Tabla 2.2 se aplica a los que se han codificado mediante intracodificación

y la Tabla 2.3 a los que se han codificado mediante intercodificación. Aún así estas son las tablas *predeterminadas*, ya que el formato nos permite especificar nuestra propia tabla de cuantificación.

Las tablas de cuantificación ($W[v][u]$) servirán para cuantificar los coeficientes AC[1] según la expresión (donde, recordemos, Q_p representa a la variable `quantiser_scale`.).

$$QF[v][u] = (F[v][u] \cdot 16 // W[v][u] - k \cdot Q_p) // (2 \cdot Q_p) \tag{2.3}$$

Donde cumplimos:

$$k = \begin{cases} 0 & \text{Para macrobloques intracodificados} \\ sign(QF[v][u]) & \text{Para macrobloques intercodificados} \end{cases} \tag{2.4}$$

Para la operación inversa utilizamos las mismas tablas de cuantificación y la expresión relacionada es:

$$F^{'}[v][u] = \begin{cases} 0 & \text{Para} QF[v][u] = 0 \\ ((2 \cdot QF[v][u] + k) \cdot W[v][u] \cdot Q_p)/16 & \text{Para} QF[v][i] \neq 0 \end{cases} \tag{2.5}$$

Donde el operador / indica una división entera con truncamiento y debemos cumplir la misma condición (2.4) que antes.

Segundo Método de Cuantificación: Cuantificación H.263

Este método, al contrario que el anterior, no utiliza tablas de cuantificación. Esta técnica es menos compleja y más sencilla de implementar, pero no consigue optimizar el cuantificador aprovechando la cuantificación adaptativa.

El proceso se realiza siguiendo la siguiente expresión:

$$\mid QF[v][u] \mid = \begin{cases} \mid F[v][u] \mid / (2 \cdot Q_p) & \text{Para macrobloques intracodificados} \\ (\mid F[v][u] \mid - Q_p/2)/(2 \cdot Q_p) & \text{Para macrobloques intercodificados} \end{cases} \tag{2.6}$$

Una vez obtenido $\mid QF[v][u] \mid$ obtenemos $QF[v][u]$ a partir del signo de $F[v][u]$ según la forma:

$$QF[v][u] = sign(F[v][u]) \cdot \mid QF[v][u] \mid \tag{2.7}$$

La cuantificación inversa la vemos a continuación:

[1] Recordemos que los coeficientes DC se cuantifican según vimos anteriormente

$$| F^{'}[v][u] | = \begin{cases} 0 & \text{Para} QF[v][u] = 0 \\ (2\cdot | QF[v][u] | +1)\cdot Q_p & \text{Para } QF[v][u] \neq 0 \text{ y } Q_p \text{ impar} \\ (2\cdot | QF[v][u] | +1)\cdot Q_p - 1 & \text{Para } QF[v][u] \neq 0 \text{ y } Q_p \text{ par} \end{cases} \quad (2.8)$$

Y para obtener $F^{'}[v][u]$:

$$F^{'}[v][u] = sign(QF[v][u])\cdot | F^{'}[v][u] | \quad (2.9)$$

2.3.2. Predicción de Coeficientes para Macrobloques Intracodificados: Predicción AC/DC

Para la mayoría de coeficientes AC y DC de bloques vecinos existen relaciones de dependencia. De esta forma la energía media de los coeficientes cuantificados puede reducirse, aún más, utilizando predicción con los bloques vecinos.

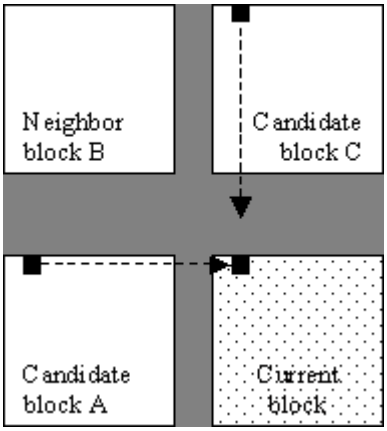


Figura 2.4: Bloques candidatos para la predicción de coeficientes.

La predicción puede realizarse con los bloques que vemos en la Figura 2.4. La dirección de la predicción es adaptativa y es seleccionada basándonos en la comparación de los gradientes horizontales y verticales del DC de los bloques A, B y C.

Hay dos tipos de predicción posibles:

- **Predicción DC**: La predicción se realiza para el coeficiente DC, escogiendo entre el DC de A o de C.

- **Predicción AC**: Los coeficientes de la primera columna y la primera fila del bloque actual se predicen con los coeficientes situados en la misma posición de los bloques candidatos (A y C).

Si alguno de los bloques (A, B o C) se encuentran fuera del VOP correspondiente al bloque actual se asume que los coeficientes DC de los mismos toman el valor $2^{bits\_per\_pixel+2}$.

2.3.3. Codificación Entrópica

Después de la predicción AC/DC se aplica una codificación entrópica sobre los coeficientes de la DCT, usando una tabla de código variable. Pero antes, debemos transformar la matriz $PQF[v][u]$ (resultado de realizar la predicción sobre la matriz cuantificada) en un vector unidimensional $QFS[n]$ que será sobre el que reduciremos la entropía.

Tabla 2.4: Tabla de recorrido en Zig-Zag.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Hay tres formas de realizar esta transformación previa:

1. **Recorrido en Zig-Zag**: Al estilo del codificador JPEG la matriz es recorrida en Zig-Zag y los coeficientes sobre los que vamos pasando van añadiéndose al vector $QFS[n]$ (Tabla 2.4).

2. **Recorrido Horizontal**: Los coeficientes son recorridos horizontalmente y añadiéndose a $QFS[n]$ de la misma forma que en el caso anterior (Tabla 2.5).

3. **Recorrido Vertical**: Igual que en el recorrido horizontal, salvo que el movimiento se realiza verticalmente (Tabla 2.6).

Tabla 2.5: Tabla de recorrido horizontal.

0	1	2	3	10	11	12	13
4	5	8	9	17	16	15	14
6	7	19	18	26	27	28	29
20	21	24	25	30	31	32	33
22	23	34	35	42	43	44	45
36	37	40	41	46	47	48	49
38	39	50	51	56	57	58	59
52	53	54	55	60	61	62	63

Tabla 2.6: Tabla de recorrido vertical.

0	4	6	20	22	36	38	52
1	5	7	21	23	37	39	53
2	8	19	24	34	40	50	54
3	9	18	25	35	41	51	55
10	17	26	30	42	46	56	60
11	16	27	31	43	47	57	61
12	15	28	32	44	48	58	62
13	14	29	33	45	49	59	63

La finalidad de esta operación es que los coeficientes de mayor valor espectral se sitúen en las primeras posiciones del vector, mientras que los coeficientes con valor 0 se colocarán en las últimas, permitiendo una mayor reducción de la entropía.

La elección de una u otra tabla se rige de la siguiente forma:

- Si no se ha utilizado la predicción AC/DC para un bloque intracodificado se utiliza la tabla en Zig-Zag.
- Si se ha utilizado predicción AC/DC la dirección de predicción del coeficiente DC será la que seleccione la tabla. Si el coeficiente DC que se tomó correspondía con el bloque superior al de referencia se utilizará la tabla de recorrido horizontal. Si, por el contrario, el bloque del que se tomó el coeficiente DC era el de la izquierda se tomará la tabla de recorrido vertical.

- Si el bloque es un bloque intercodificado se aplicará la tabla en Zig-Zag.

Una vez tenemos el vector $QFS[n]$ listo y ordenado se aplica sobre éste un código RLE (*Run Lenght Encoding*). Éste viene a ser una solución elegante y óptima para eliminar la redundancia dentro del vector, dado que constará de un número significativo de *ceros* en las últimas posiciones del mismo.

2.3.4. Herramientas para la Compensación del Movimiento

Justo antes de aplicar el código de longitud variable se aplican las operaciones inversas a la cuantificación y a la DCT descritas anteriormente. El resultado es almacenado con la finalidad de ser utilizado para la compensación y estimación de movimiento (Motion Compensation y Motion Estimation) en el proceso de predicción.

En caso de ser un P-VOP, el reconocido por el codificador, se realiza una compensación de movimiento sobre cada macrobloque. La diferencia espacial encontrada entre el P-VOP y el VOP de referencia es almacenada en un vector de movimiento, sobre el que se efectuará la DCT y la cuantificación (dividiéndolo, nuevamente, en macrobloques).

Nuevamente se vuelve a regenerar el P-VOP aplicando las funciones inversas y se almacena para que sirva de VOP de referencia para otros posibles P-VOPs o B-VOPs.

El proceso para los B-VOPs es muy similar al de los P-VOPs, salvo que la compensación de movimiento se realiza de forma bidireccional y no es almacenado, ya que no se utilizan como VOPs de referencia.

Antes de pasar a enumerar los algoritmos básicos del proceso de compensación de movimiento veremos cómo puede representarse la información de movimiento. Existen dos modos básicos definidos a niveles de macrobloque.

1. **Modo 1MV**: Un vector de movimiento (MV) es transmitido para el macrobloque completo.

2. **Modo 4MV**: El macrobloque es dividido en 4 bloques de 8x8 y un MV es transmitido para cada uno de ellos.

La compensación de movimiento cuenta con 3 algoritmos básicos:

1. **Compensación de movimiento Quarter-pel**: Utiliza vectores de movimiento con una resolución incrementada a un cuarto de pixel, en contra de las resoluciones de medio pixel y pixel completo que son utilizadas en H.261, H.263, MPEG-1 y MPEG-2. Esto supone una mejora en el proceso de predicción y una disminución del error asociado.

2. **Compensación de movimiento global**: Se cuenta con una información de movimiento unificada para cada VOP. Esta técnica es de especial interés en movimientos rápidos de cámara y efectos como el *zoom*, donde cada parte de la imagen recibe una traslación espacial idéntica.

3. **Modo directo en la predicción bidireccional**: Es una mejora de la predicción de compensación de movimiento bidireccional utilizada para codificar los B-VOPs a partir de los P-VOPs adyacentes. Es una generalización de la técnica introducida por H.263.

Antes de analizar en profundidad estos algoritmos echaremos un vistazo a la estructura básica que nos permitirá entenderlos correctamente.

Estructura de Predicción Temporal

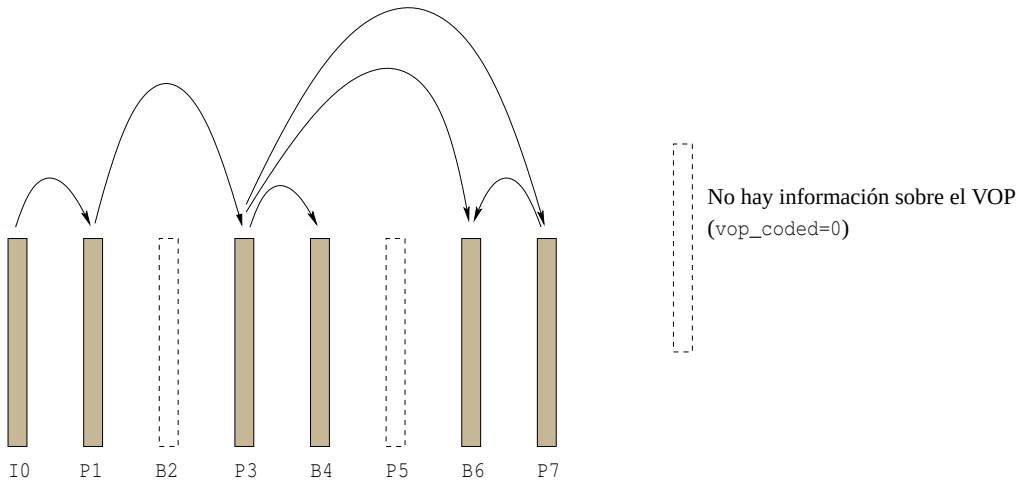


Figura 2.5: Estructura de predicción temporal.

Tal y como vemos en la Figura 2.5 los I-VOP no tienen ninguna dependencia con ningún otro VOP. Los P-VOP tienen dependencias con el I-VOP o P-VOP inmediatamente anterior, en el caso que su información haya sido transmitida y, por tanto, `vop_coded=1`. Por último los B-VOP tienen dependencia bidireccional con los I-VOP o P-VOP inmediatamente anterior y posterior.

Es importante comprender que la Figura 2.5 representa el orden de visualización y no el de transmisión. Como los B-VOP tienen dependencias con respecto a VOPs que

deberán ser visualizados en un futuro éstos son transmitidos antes que el B-VOP correspondiente. Para el ejemplo el orden de transmisión sería: `I0, P1, P3, B2, P5, B4, P7, B6`, de ésta forma todas las dependencias serían resueltas. Sin embargo no hay información sobre los VOPs `B5` y `P5` en el momento actual (no han sido transmitidos), lo que supone que `B6` tenga que recurrir a la información contenida en `P3`, al igual que `P7`.El que no haya información sobre `B2` no es relevante, dado que ningún VOP depende de un B-VOP.

Compensación de Movimiento Quarter-Pel

Los vectores de movimiento utilizados para la compensación de movimiento deben ser transmitidos al descodificador. De esta forma, la resolución queda limitada a un número finito.

Como el movimiento real entre dos imágenes sucesivas de un secuencia no queda definido correctamente por un pixel (lo que supone un error considerable en la predicción) el MPEG-4 utiliza un cuarto del mismo. De esta forma, aumentando la resolución, se consiguen mejoras en la predicción y un error menos significativo.

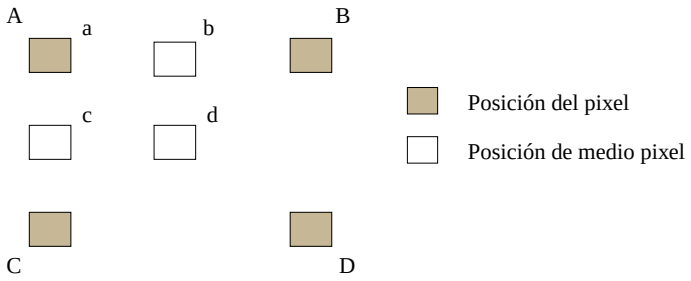


Figura 2.6: Interpolación de las muestras para una resolución de medio pixel utilizando interpolación bilineal.

Como vemos en la Figura 2.6 los vectores de movimiento con resolución de medio pixel pueden colocarse de varias formas, provocando comportamiento distintos por parte del predictor de compensación de movimiento.

Para el caso de la posición `a`, las dos componentes del vector de movimiento son de pixel completo por lo que no se realiza interpolación. Para las posiciones `b` y `c` sólo una de las componentes tiene resolución de medio pixel, por lo que el valor de la muestra es calculado por interpolación bilineal de las muestras vecinas. Por tanto, $b = (A+B+1)/2$ y $c = (A+C+1)/2$. Para la posición `d`, que tiene las dos componentes con resolución de medio pixel la interpolación resultaría de: $d = (A+B+C+D+2)/4$.

Para una resolución de un cuarto de pixel no podemos seguir el mismo método. Un incremento de la resolución no implicaría una reducción del error significativa, mientras que el bitrate para la estimación de movimiento debería aumentarse, con la consecuente pérdida de compresión.

Es por ello que el MPEG-4 implementa el algoritmo en dos fases:

En la primera de ellas las muestras de resolución de medio pixel son calculadas utilizando una interpolación mejorada. Ahora no tendremos en cuenta sólo a 2 ó 4 de las muestras vecinas, sino que utilizaremos 8 muestras en cada dirección. Para la interpolación se utilizará un filtro de respuesta de impulso finita (FIR), cuyos valores se encuentran especificados en el *MPEG-4 Visual Standard* y son los siguientes:

$$[-8/256, 24/256, -48/256, 160/256, 160/256, -48/256, 24/256, -8/256] \quad (2.10)$$

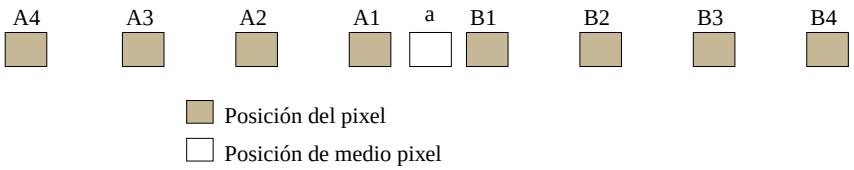


Figura 2.7: Interpolación de las muestras para una resolución de medio pixel utilizando interpolación mejorada.

Tomemos como ejemplo la Figura 2.7. En este caso el valor de la muestra a sería:

$$a = (-8 \cdot A4 + 24 \cdot A3 - 48 \cdot A2 + 160 \cdot A1 + 160 \cdot B1 - 48 \cdot B2 + 24 \cdot B3 - 8 \cdot B4)/256 \quad (2.11)$$

Para realizar el cálculo de la posición vertical el procedimiento sería idéntico, salvo porque las muestras de referencia serían las ubicadas verticalmente. Si nos encontramos en la situación en que las dos componentes del MV son de medio pixel se aplicaría primero el filtrado horizontal y luego el vertical.

En el segundo paso del algoritmo los valores para los cuartos de pixel son calculados a partir de los valores de medio pixel hallados utilizando la interpolación FIR. En esta ocasión el método a utilizar sí es la interpolación bilineal que vimos anteriormente.

Compensación de Movimiento Global

Esta técnica no se suele aplicar sobre un VOP al completo, ya que suele generar errores de predicción significativos en algunas áreas. Por tanto, aunque la información de movi-

miento viene incluida en el VOP y es única, ésta se suele aplicar sobre los macrobloques del mismo de forma individual.

Como decíamos, la información de movimiento se encuentra incluida en la cabecera del VOP, pero el codificador deberá decidir si, para cada macrobloque, utilizará la compensación de movimiento global (GMC) o local. Por tanto, cada macrobloque también incluirá información sobre el tipo de compensación con la que fue codificado.

La información de la que hablamos incluye hasta 4 vectores de movimiento para el VOP, así como sus posiciones referencia. Éstos 4 vectores serán utilizados para la compensación de movimiento de los macrobloques. Para cada pixel de éstos se calcula un vector de referencia basándonos en los vectores de referencia del VOP y su posición de referencia (se realiza una especie de interpolación para cada posición del macrobloque). La resolución de esos MV queda especificada por el codificador y guardada como información para la posterior descodificación. En caso de tener una resolución subpixel los valores de las muestras serán calculados mediante interpolación bilineal.

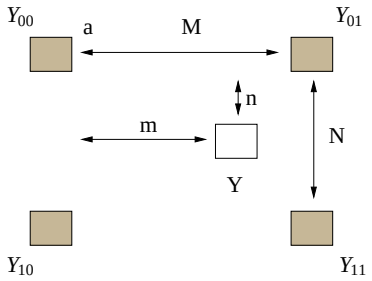


Figura 2.8: Interpolación de las muestras para una resolución de subpixel en GMC.

Si tomamos como ejemplo la Figura 2.8 donde Y_{00}, Y_{01}, Y_{10} y Y_{11} son los valores de las muestras del VOP de referencia el valor de predicción Y resulta de:

$$Y = \frac{N-n}{N} \cdot \frac{N-m}{M} \cdot Y_{00} + \frac{N-n}{N} \cdot \frac{m}{M} \cdot Y_{01} + \frac{n}{N} \cdot \frac{M-m}{M} \cdot Y_{10} + \frac{n}{N} \cdot \frac{m}{M} \cdot Y_{11} \qquad (2.12)$$

Modo Directo en la Predicción Bidireccional

Para la compensación de movimiento bidireccional se reconocen 4 modos distintos de predicción:

1. **Forward Mode**: Sólo se tiene en cuenta la referencia anterior (o referencia hacia

delante [2]). Sólo se transmite un MV.

2. **Backward Mode**: Sólo se tiene en cuenta la referencia posterior. Igualmente sólo se transmite un MV.

3. **Interpolative Mode**: Se tienen en cuenta las dos referencias y se transmiten dos MV. La predicción resultante se obtiene de la interpolación de los valores de referencia.

4. **Direct Mode**: Como en el caso anterior se tienen en cuenta las dos referencias, sin embargo los vectores de movimiento se obtienen a partir de los MV del macrobloque de la referencia posterior (o de referencia hacia atrás). Sólo se transmite el `delta vector` (la corrección).

Los tres primeros modos se utilizan en el estándar MPEG-2, mientras que el último se ha creado especialmente para el MPEG-4.

La idea básica de este método es la de explotar el conocimiento de los vectores de movimiento entre los VOP de referencia anterior y posterior al VOP actual.

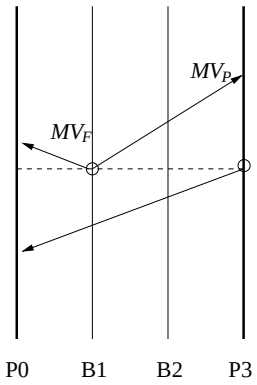


Figura 2.9: Los vectores MV_F y MV_P se derivan del MV de la referencia hacia atrás con respecto a la referencia hacia delante.

Tal y como vemos en la Figura 2.9 para hallar los MVs del VOP actual se utiliza el MV de la referencia posterior con respecto a la referencia anterior. Para ello este último es escalado, dependiendo de la posición en el tiempo del B-VOP en cuestión entre los

[2]En este apartado nos referiremos indistintamente a la referencia posterior como *de referencia hacia atrás* y a la referencia anterior como *de referencia hacia delante*, ya que es así como se suelen denominar en la mayoría de artículos técnicos para explicar el desarrollo de este método.

dos VOPs de referencia. A continuación un término de corrección denominado `delta` es añadido al resultado. Éste será precisamente el término que se transmitirá junto al macrobloque del B-VOP para la etapa de descodificación.

Los factores de escala se calculan de la siguiente forma:

$$TRB = \texttt{display\_time} \cdot (VOP\_Actual) - \texttt{display\_time} \cdot (referencia\_posterior) \tag{2.13}$$

$$TRV = \texttt{display\_time}(referencia\_posterior) - \texttt{display\_time}(referenciaanterior) \tag{2.14}$$

Ahora realizamos el escalado y la suma del término de corrección `delta` (MV_D):

$$MV_F = \frac{TRB}{TRD} \cdot MV + MV_D \tag{2.15}$$

$$MV_B = \frac{TRB - TRD}{TRD} \cdot MV + MV_D \tag{2.16}$$

MV_F se refiere al vector de movimiento de referencia hacia delante (es decir, con el macrobloque de referencia anterior) y MV_B al vector de movimiento de referencia hacia atrás (con el macrobloque de referencia posterior).

En el caso de que los macrobloques se hallan codificado en el modo de 4MVs por bloque se realizará esta operación 4 veces, una para cada bloque de 8x8 y para su respectivo MV. Es más, este es el único método que permite realizar estimación de movimiento bidireccional sobre macrobloques codificados en modo de 4MVs.

Finalmente, indicar que esta técnica puede provocar errores en la predicción. En el caso de no existir el macrobloque de referencia del I-VOP o P-VOP de referencia hacia atrás el macrobloque actual del B-VOP realiza la corrección con vectores de movimiento nulos, al no *esperar* a la descodificación del próximo VOP que le sirva de referencia.

2.4. Resistencia a Fallos

La compresión de la información implica la eliminación de redundancia, haciendo más difícil la reconstrucción de la información en caso de error.

Pese a ser un algoritmo menos sensible a los posibles errores que el MPEG-2, debido entre otros factores a su bajo bitrate (a menor número de bits menor posibilidad de que uno de ellos se transmita incorrectamente), hay otras características que le obligan a utilizar sistemas complejos para la detección de los mismos.

Este algoritmo utiliza un código de longitud variable, por lo que un fallo en la transmisión puede suponer que el descodificador determine incorrectamente la longitud de una palabra del código y se pierda la sincronización entre éste y el flujo de datos.

Además la codificación predictiva de los VOPs permite *propagar* un error en la transmisión hasta la llegada de un VOP completo (I-VOP).

Un caso real donde podemos ver los inconvenientes de este método de codificación son las retransmisiones de eventos deportivos, como podría ser la fórmula 1. Las cámaras que se encuentran dentro de los monoplazas transmiten los datos a los servidores centrales utilizando algoritmos de codificación predictiva, consiguiendo ahorrar ancho de banda en la comunicación. Sin embargo, según las condiciones atmosféricas, (ya que las transmisiones se realizan por ondas electromagnéticas utilizando tecnología Wireless) se dan situaciones donde la imagen se distorsiona durante varios segundos, aunque realmente el error no se ha repetido durante todo el intervalo de tiempo, sino que se ha propagado debido a la codificación incremental de los *frames*.

Es por ésto que el MPEG-4 permite especificar la frecuencia de I-VOPs dentro una trama de datos, cuantos más tengamos mayor ancho de banda necesitaremos para transmitir, ya que la cantidad de datos será mayor. Si por el contrario optamos por número de I-VOPs muy bajo obtendremos un mayor tasa de compresión, pero en caso de obtener un error en la transmisión éste se propagará durante más tiempo.

Una de las herramientas que utiliza el MPEG-4 para aumentar la resistencia a fallos es un bloque de detección de error situado en el descodificador que es capaz de detectar datos no válidos, fuera de rango o un exceso de los mismos.

Para *intentar* solucionar el problema que supone utilizar un código de longitud variable se utiliza un sistema de código reversible. De esta forma si se detecta un error en un bit se puede seguir descodificando en dirección inversa.

2.4.1. Localización del Error

El estándar MPEG-4 utiliza cabeceras para marcar el comienzo de las estructuras de datos más importantes. Estas cabeceras utilizan 32 bits y nos permiten, por ejemplo, conocer el tipo de objeto que viene a continuación y el comienzo de los VOPs dentro del flujo de datos.

Hay una herramienta para la localización de errores conocida como *Video Packet Mode* que, basándose en las cabeceras, permite segmentar el flujo de datos en paquetes (en el proceso de codificación). Posteriormente en la descodificación y gracias a un sistema de sincronización implementado por el codificador se puede reconstruir la imagen completa colocando cada paquete en su sitio. Sin embargo, en caso de que el error se produzca en el sistema de sincronización se podrían perder todos los paquetes restantes de la imagen actual.

Cada uno de los paquetes es codificado de forma independiente, lo que implica que no existen dependencias entre ellos y que en caso de error éste no se propagará (los vectores de movimiento son independientes). Además, las cabeceras de los mismos registran in-

formación sobre el proceso de cuantificación, las posiciones de los macrobloques, etc., lo que permite reconstruir la cabecera del VOP al completo en caso de que haya un error en la misma (hay redundancia de información, lo que implica una menor compresión, pero es necesaria para disminuir la tasa de errores).

Cuando uno de los paquetes es marcado como *erróneo* el descodificador revisa el contenido de los frames tanto anterior como posterior basándose en el sistema de sincronización por marcos, lo que le permite ajustar el motivo del error, así como su naturaleza.

En la Figura 2.10 podemos ver un ejemplo del funcionamiento del *Video Packet Mode* y su respuesta ante un error en uno de los paquetes.

(a) Imagen original

(b) Imagen segmentada en paquetes

(c) Error localizado en uno de los paquetes

(d) Se reconstruye la imagen a excepción del paquete marcado

Figura 2.10: Funcionamiento del *Video Packet Mode*.

Capítulo 3

Implementación

3.1. Introducción

En la parte práctica del proyecto se han implementado dos elementos software, utilizando para ello el lenguaje de programación C++. En un principio el lenguaje elegido fue C, pero debido a la necesidad de utilizar la biblioteca `avifile` [9] (programada en C++) se realizó el cambio (aunque la mayoría del código sigue siendo C estándar, a excepción de la parte relacionada con la biblioteca en cuestión).

El servidor será el encargado de enviar las tramas MPEG-4 a través de un canal Multicast, para lo que deberá leer cierta información de la cabecera del mismo (utilizando la biblioteca `avifile`). Esta información será necesaria para la posterior descodificación por parte del cliente.

Por otra parte encontramos al cliente, que será el encargado de recibir los datos accediendo al servidor, descodificarlos (utilizando una versión modificada de la biblioteca `FFMPEG` [10]) y mostrar las imágenes generadas por pantalla (utilizando la biblioteca `SDL`, [11]).

En el proceso de desarrollo se han encontrado ciertas dificultades, la mayoría de ellas relacionadas con la estructura de las tramas de datos MPEG-4. A pesar de ser un sistema de codificación pensado para la transmisión en redes de difusión, no cuenta con descodificadores que exploten todas las características del mismo (la biblioteca utilizadas es, a día de hoy, una de la mejores implementaciones del estándar en cuanto a eficiencia y a aprovechamiento de las funcionalidades del mismo).

En las secciones venideras se realizará un análisis del procedimiento llevado a cabo para la implementación del software en cuestión, haciendo especial hincapié en los problemas encontrados y cómo se han tratado de solucionar.

3.2. El Servidor

Nuestro servidor es el elemento más básico de nuestra implementación. Su misión será la de establecer el canal Multicast por el que transmitirán las tramas de datos.

Cuando iniciamos la ejecución el programa inicia dos hilos independientes. El primero de ellos será encargado de leer la cabecera del archivo seleccionado, extrayendo de la misma cierta información que será enviada a los clientes posteriormente (tamaño en pixels de la imagen, bitrate, frames por segundo, etc.). Cuando toda la información esté disponible iniciará el canal Multicast, creará una señal de interrupción del sistema, y se mantendrá en espera de peticiones de conexión.

La señal de interrupción llamará a una función encargada de enviar los datos al canal en intervalos de tiempo fijos, establecidos según el tamaño del buffer y el bitrate de la secuencia de vídeo (con la finalidad de que el envío de datos no sea excesivamente rápido y el cliente tenga tiempo de recogerlos y descodificar el vídeo de forma eficiente).

El segundo hilo será el encargado de iniciar una conexión TCP cada vez que llegue una petición por parte de un cliente. Cuando se realice la conexión el hilo será lanzado y enviará los datos de la cabecera, el tamaño del buffer, la dirección Multicast y puerto al que debe conectar, así como la primera trama de datos al cliente.

El porqué de utilizar una conexión TCP en un sistema Multicast es el siguiente: debemos asegurar que el cliente recoja los datos iniciales de información, así como la primera trama de datos.

Lo primero es claro, el cliente debe conocer datos como el tamaño del vídeo o la dirección del canal Multicast al que realizar la conexión (es mucho más sencillo que el cliente se conecte a un servidor del que conoce su nombre de dominio o su dirección IP que tener que informarse del canal Multicast que se encuentra emitiendo en ese momento).

Pero el recibir la primera trama de datos es, si cabe, aún más importante. Recordemos que el descodificador de MPEG-4 necesita un I-VOP inicialmente para poder realizar la compensación y estimación de movimiento en el proceso de predicción para el siguiente VOP. En caso de no encontrar un I-VOP inicialmente el descodificador no es capaz de realizar el proceso, ignora si el fallo es debido a un archivo corrupto, una mala transmisión o si simplemente no es un vídeo en formato MPEG-4, y termina la ejecución. Como el cliente puede llegar en cualquier momento no podemos asegurar que lo que recibe inicialmente es, en efecto, un I-VOP. Por tanto, si enviamos el primer segmento de datos aseguramos que lo primero que recibirá será un I-VOP. El efecto es prácticamente inapreciable, dado que este primer segmento de datos equivale a menos de un segundo de la secuencia de vídeo. Otra posible solución habría sido *buscar* el primer I-VOP de la secuencia recogida del canal por el cliente y empezar a enviar datos al descodificador a partir de ahí, pero esto presenta inconvenientes que van en contra del concepto inicial del proyecto: por una parte la sincronización sería más lenta, es más rápido descodificar me-

nos de un segundo de vídeo que buscar dentro del buffer el primer I-VOP. Por otro lado, podría no encontrarse ningún I-VOP dentro del mismo y tendríamos que volver a recoger datos del canal, perdiendo los que actualmente tenemos y, por tanto, parte de la secuencia de vídeo, cuando con la primera solución empezaríamos la visualización de forma inmediata. Concluyendo, la solución elegida es la que más se acerca a nuestro cometido: realizar un sistema lo más eficiente posible.

Otro problema menor es el de seleccionar el tamaño del buffer. Ya que los mensajes UDP no son fragmentables de forma automática (al contrario que el TCP el fragmentado de mensajes debe realizarlo el programador) y que encontramos un límite del tamaño de trama establecido en aproximadamente 64 KB (el IP reserva 16 bits para especificar el tamaño del mensaje, su cabecera requiere de 24 Bytes y la del UDP de 8 Bytes por tanto, $2^{16} - 24 - 8 = 65504$ Bytes quedan disponibles para la parte de datos). Por tanto no podemos limitarnos a enviar un buffer de tamaño igual al bitrate de la secuencia cada segundo, debemos comprobar si éste valor supera estos 65504 Bytes para, en caso afirmativo, fragmentarlo y enviar en periodos de tiempo más distanciados. Un ejemplo de ésto sería un vídeo con un bitrate de 800 kbs[1], que equivale a 100 kBs, enviar 100 kB cada segundo no sería posible ya que supera el límite establecido ($100kB = 97{,}65KB$) , en este caso el buffer podría ser de 50 kB y transmitir cada 0.5 segundos, de esta forma el efecto es parecido y es posible realizar la comunicación. Decimos que el efecto es parecido porque para el cliente no es lo mismo interrumpir el proceso de descodificación y generación de la secuencia de vídeo el doble de veces. Aunque el mensaje sea la mitad de grande ésto no supone un trabajo computacional extra significativo (el trabajo es realizado por la interfaz de red), pero *parar* todo el trabajo por medio de la señal de interrupción para atender la comunicación sí lo es.

3.3. El Cliente

La parte del cliente es, sin lugar a dudas, la parte crítica de nuestro proyecto. En ella se debe:

1. Recoger el flujo de datos del canal Multicast (se ha mostrado especial interés en ésto, se podría llegar incluso a detener los demás procesos para evitar la pérdida de datos).

2. Descodificar los datos obtenidos del canal, de la forma más eficiente posible y sin que ésto ocasione un cuello de botella para el proceso general.

[1] Con 800 kbs nos referimos a $800 \cdot 10^3 bits/s$, ya que el bitrate se suele medir de esta forma, y con 64 KB a $64 \cdot 2^{10} Bytes$.

3. Mostrar las imágenes en pantalla, lo que implica una atención especial, ya que tienen que mostrarse a intervalos fijos, no como la descodificación o la recogida de datos, que podrían llegar a esperar en determinados momentos.

La mayor parte del tiempo de trabajo se ha invertido en realizar la sincronización de los tres procesos, ya que se intenta que en ningún caso se tengan que esperar unos a otros.

Se utilizan dos buffers circulares, ambos dinámicos, creados a partir del bitrate del vídeo. Uno de ellos será el que se encargue de la recogida de datos y el otro estará destinado a conservar las imágenes ya descodificadas para ir mostrándolas por pantalla.

El funcionamiento del descodificador es bastante básico: a partir de un puntero a memoria introducido como parámetro éste realiza las operaciones convenientes para descodificar el siguiente frame. El resultado es el número de Bytes utilizados en el proceso y el frame descodificado en un espacio de color determinado (RGB, YUV420, etc.). Para que la visualización por pantalla sea correcta debemos realizar la transformación del resultado al espacio YUV420 (ver Apéndice B).

Se ha intentado no hacer suposiciones sobre la velocidad de los procesos, de forma que se han establecido *situaciones límite* donde nuestro sistema podría *agotar* alguno de esos buffers, con la consecuente pérdida de eficiencia.

Si en algún momento hay un fallo en la conexión o la descodificación es mucho más rápida que la recogida de datos (situación que podría deberse a una mala elección del tamaño del buffer) el sistema se *congela*. La finalidad es evitar que el puntero que señala la siguiente posición del buffer por la que comenzar la descodificación *supere* al que señala la posición a sobreescribir en la recogida de datos. Esto no sería nada deseable, pues comenzaríamos a mostrar imágenes anteriores y perderíamos la sincronización. Nuestra solución es sencilla: el proceso de descodificación se detiene y, para evitar que el buffer de imágenes descodificadas se agote los punteros no se modifican, con lo que la imagen permanece estática. Cuando la señal vuelve el proceso se reinicia por donde lo dejó.

También podría darse la situación en que el proceso de descodificación fuese demasiado lento, con la consecuente sobreescritura de los datos ya obtenidos, lo que provocaría una pérdida de los mismos de forma irreversible y posibles fallos en el proceso de descodificación. Por eso, de darse esa situación el sistema lo detecta y deja de recoger datos del canal. También perderemos datos, pero el proceso de descodificación estará libre de errores. De todas formas, de ocurrir, lo más seguro es que el equipo simplemente no sea capaz de realizar la descodificación en tiempo real, ya que sin lugar a dudas es el proceso más costoso de cuantos realiza el sistema. Un ejemplo gráfico del funcionamiento de los punteros se puede ver en la Figura 3.1.

Otra situación, prácticamente fuera de nuestro control, sería que el proceso de presentación por pantalla de las distintas imágenes fuese demasiado lento (debido a una tarjeta gráfica de bajas prestaciones, por ejemplo) situación que supondría una parada irreversible en todo el sistema. Ésto es debido a que la visualización por pantalla es realizado

por una función que es llamada a intervalos de tiempo constantes (asegurando de esta forma el número de imágenes por segundo necesario), utilizando una señal de interrupción del sistema. Cuando una señal se activa todos los procesos se paran, la función realiza las operaciones convenientes y cuando ha terminado vuelve a reiniciarlo todo. Así, una parada excesiva, puede suponer una pérdida de la sincronización.

Inicialmente esta señal iba a implementarse utilizando la biblioteca `signal.h` de C, pero en las primeras pruebas de nuestro programa detectamos un comportamiento anómalo. Las señales creadas con esta biblioteca llevan asociada una función y un temporizador, que es el que representa el tiempo *máximo* que deberá pasar antes de llamar a la rutina asociada. Decimos tiempo máximo porque en caso de producirse una señal de sistema (como las que generan las `XFree86` en ciertas situaciones) el temporizador expira de forma instantánea. Ésto provocaba que el proceso de visualización fuese excesivamente rápido bajo determinadas circunstancias, ya que el temporizador expiraba antes del tiempo establecido. Ante esta situación decidimos utilizar una función de la biblioteca `SDL`, de funcionamiento parecido pero que no se ve afectada por la señales de sistema.

Otro problema, ajeno a nuestra implementación, era provocado por el descodificador. La biblioteca `FFMPEG` está pensada para secuencias de datos *carentes de errores*, es decir, datos recogidos directamente de un disco duro o de una conexión TCP, pero no de una transmisión Multicast donde éstos pueden estar desordenados o, incluso, podemos perder parte de los mismos. El efecto no era nada deseable: cuando una secuencia de datos era *corrupta* y el descodificador encontraba algún problema detectaba el fallo y terminaba (o hacía una operación de referencia ilegal a memoria). Por tanto, se decidió modificar el código fuente, buscando situaciones de comprobación de la validez de los datos para modificarlas o, simplemente, eliminarlas. El reconocimiento de errores es una herramienta muy útil, pero para cometidos como el nuestro la existencia de error no debe suponer el fin de la ejecución.

En la Figura 3.2 podemos ver un diagrama de bloques con el esquema de funcionamiento del sistema al completo. Como vemos, tras una conexión TCP son realizadas distintas operaciones relacionadas entre sí (recogida de datos, descodificación y visualización de las imágenes). Éstas comparten buffers circulares entre sí tal y como describimos anteriormente, de forma que el resultado de la recogida de datos es introducido en el descodificador y el resultado del mismo es tratado para su posterior visualización.

3.4. Ejecución

Para la ejecución del sistema debemos tener un vídeo en formato MPEG-4, con encapsulado AVI, almacenado localmente en el servidor. Para lanzarlo ejecutaremos, en la `shell`, el comando:

```
./server <archivo_MPEG-4> <dirección_multicast> <puerto> [TTL]
```

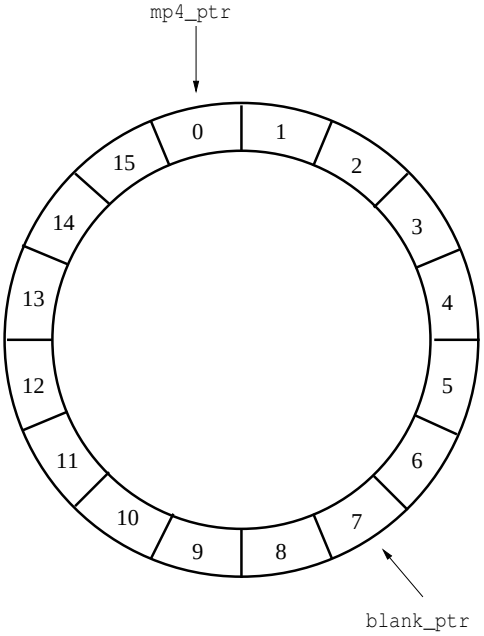
Posteriormente podremos lanzar el proceso *cliente*, mediante el comando:

```
./client <ip_servidor>
```

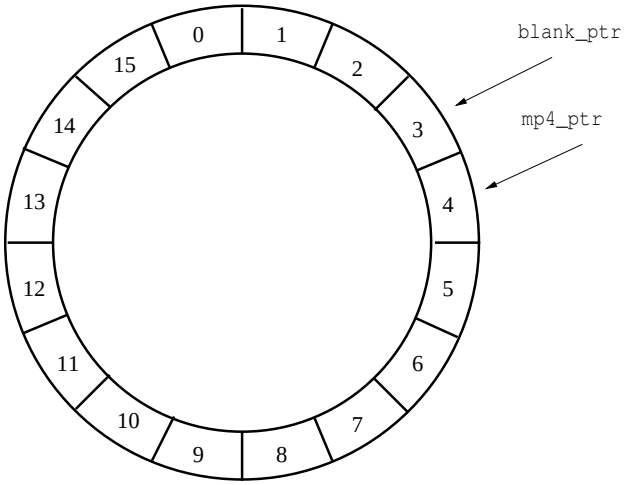
Es necesario que el equipo desde el que se lanza el proceso cliente se encuentre dentro de la misma LAN que el servidor. En caso contrario, tal y como vimos en el capítulo dedicado a la transmisión, necesitaremos routers Multicast a ambos lados (del cliente y del servidor), para que el *tunneling* pueda ser llevado a cabo y el encapsulamiento IP-in-IP permita la transmisión de los datos.

Si nos encontramos en el lado del cliente, y la conexión se ha realizado correctamente, aparecerá una ventana con el tamaño del vídeo en cuestión. Ésta es controlada mediante eventos, de forma que puede redimensionarse la imagen (manteniendo la relación entre el ancho y alto, o *aspect ratio*), salir del programa (pulsando la tecla `q`) o pasar a modo de pantalla completa (tecla `f`).

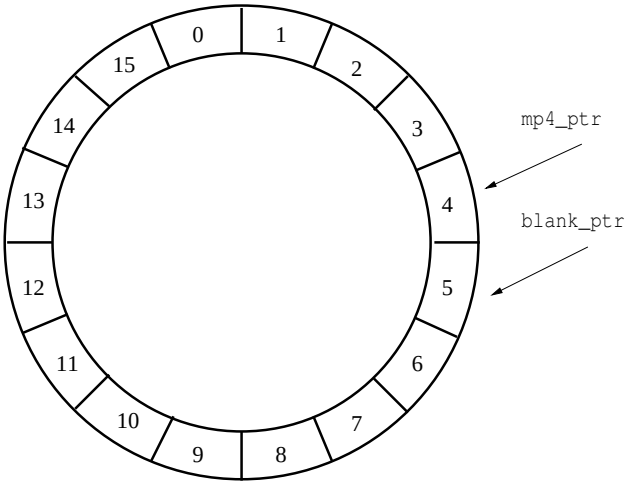
En caso de producirse errores en la transmisión la imagen se congelará o se mostrará una pantalla verde (según la naturaleza del error). Cuando el sistema vuelva a sincronizar la ejecución seguirá su curso de forma transparente para el usuario.



(a) Situación inicial, `blank_ptr` se encuentra delante de `mp4_ptr` porque ya se han recogido datos.



(b) Situación de *peligro*, la descodificación es demasiado lenta y `blank_ptr` va a rebasar a `mp4_ptr`.



(c) Situación no deseada, `blank_ptr` sobreescribe datos que aún no se han descodificado.

Figura 3.1: Respuesta del buffer circular ante una situación no deseada.

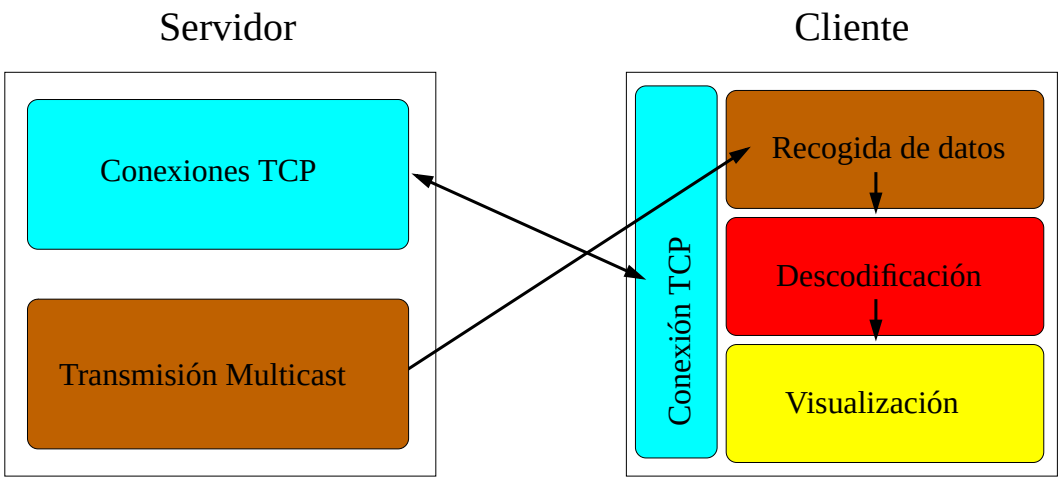


Figura 3.2: Funcionamiento del sistema al completo.

Evaluación

Para evaluar el correcto funcionamiento del sistema se ha utilizado el material descrito en el Apéndice C.

Nuestra misión era la de realizar un elemento software eficiente, que aprovechase la capacidad de una red de comunicación y resolviese los problemas de saturación del lado del servidor que ocasionan las transmisiones TCP.

A pesar de existir lenguajes de programación y librerías que ya realizan las operaciones anteriormente descritas, optamos por realizar una implementación desde la base. De esta forma conseguimos una ejecución *asíncrona*, donde los distintos procesos (a pesar de encontrarse íntimamente relacionados) se ejecutan de la forma más independiente posible.

Sin embargo, el algoritmo de descodificación MPEG-4 es bastante costoso, computacionalmente hablando, y es, sin lugar a dudas, el posible cuello de botella de nuestro sistema. Es necesaria una potencia de cálculo considerable para obtener buenos resultados, de igual forma que no todos los equipos pueden reproducir un DVD o un vídeo en formato MPEG-4 con fluidez, aunque se encuentre almacenado de forma local. Sin embargo, un buen microprocesador gráfico puede ayudar en el proceso de visualización, ayudando a que la tarea de descodificación se pueda realizar sin interrupciones.

Tabla 3.1: Algunos de los equipos utilizados para evaluar el rendimiento.

	Frec. de reloj	Mem. RAM	Mem. del mic. gráfico	C.I.E.
Intel Pentium 4	1.7 GHz	256 MB	64 MB DDR	MMX y SSE2
Intel Pentium III	450 MHz	128 MB	32 MB	MMX y SSE
Intel Celeron PIII	800 MHz	128 MB	32 MB (sin acel. 3D)	MMX y SSE

Es difícil asegurar la capacidad de cálculo mínima que debe tener un computador para poder ejecutar de forma eficiente nuestro sistema. Como ejemplo, tomemos los equipos

de la Tabla 3.1[2], que utilizamos para evaluar el rendimiento de nuestra aplicación.

Con el Pentium 4 obtuvimos unos resultados sobresalientes, todos los procesos se ejecutaban de la forma más eficiente posible. Sin embargo, pese a que el Celeron PIII tiene una frecuencia de reloj de casi el doble con respecto a la del Pentium III en éste fue imposible hacer funcionar el sistema. La carencia de aceleración 3D en su microprocesador gráfico, así como una cache L2 muy inferior a la del segundo, suponen una carga de trabajo extra para el microprocesador y una penalización por fallos de acceso a memoria que ralentizan el sistema de forma global. En el Pentium III los resultados fueron casi tan buenos como en el Pentium 4, sólo encontramos problemas trabajando con vídeos de elevado bitrate.

Por consiguiente, no es necesaria una gran frecuencia de reloj, ni un microprocesador gráfico de altas prestaciones. Lo principal es que ninguno de los elementos anteriormente citados pueda provocar un cuello de botella con respecto al resto del sistema.

[2]Donde C.I.E. representa al *Conjunto de Instrucciones Especiales*.

Conclusiones y Trabajo Futuro

Los objetivos planteados al inicio de este proyecto consistían en:

1. Realizar un estudio sobre los distintos tipos de transmisión.
2. Analizar en profundidad el sistema de codificación de vídeo MPEG-4.
3. Buscar una solución eficiente para la transmisión de vídeo a gran escala en redes poco confiables (Internet).
4. Implementar un sistema completo que nos permita obtener resultados empíricos sobre los contenidos teóricos anteriores.

Consideramos que los objetivos iniciales han sido cumplidos y que el resultado obtenido es correcto.

Por supuesto, nunca puede darse como finalizado un trabajo de esta índole, es por ello que nos gustaría señalar aquellas mejoras que consideramos interesantes y posible trabajos relacionados:

- Añadir soporte para el protocolo RTP (*Real Time Protocol*), con lo que se perdería eficiencia pero se podría aumentar la compatibilidad con el resto de aplicaciones relacionadas.
- Soporte para audio, desmultiplexando los datos antes de la descodificación y sincronizándolos en la etapa de visualización.
- Realización de un sistema de multiconferencia basándonos en los conocimientos adquiridos. En este caso se deberían tener en cuenta factores ajenos a la naturaleza de este proyecto.
- Estudio sobre distintos algoritmos de codificación y compresión, así como de sistemas reconocimiento de errores y estimación de movimiento.

- Trabajo sobre sistemas *Grid* y de multiprocesamiento, donde la elevada potencia computacional permite el desarrollo avanzado de estas áreas de conocimiento. Concretamente, nos resultaría interesante investigar sobre la paralelización de algoritmos de codificación como MPEG-4.
- Estudio sobre sistemas de transmisión progresiva de imágenes y vídeo escalable.

Apéndice A

La Transformada Discreta del Coseno (DCT)

Ya que es el mecanismo utilizado por el MPEG-4 para pasar del dominio espacial al de la frecuencia (y viceversa) los distintos VOPs que componen la secuencia de datos creemos conveniente hacer un pequeño análisis de la misma antes de continuar.

Aunque la DCT es una variación de la transformada discreta de Fourier hay una diferencia importante entre ellas y es que en la primera la imagen se descompone en sumas *sólo* de cosenos (y no de senos y cosenos como en la de Fourier).

Además, es una de las transformadas con mayores índices de compactación espectral, es decir, acumulación de la máxima cantidad de energía posible en el menor número de coeficientes[1].

La podemos definir como (A.1) en el caso de la transformación directa unidimensional y como (A.2) en el caso de la inversa (igualmente unidimensional claro está).

$$F(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos \frac{(2x+1)u\pi}{2N} \tag{A.1}$$

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) F(u) \cos \frac{(2x+1)u\pi}{2N} \tag{A.2}$$

Donde cumplimos:

$$\alpha(x) = \begin{cases} \sqrt{\frac{1}{N}} & \text{para } x = 0 \\ \sqrt{\frac{2}{N}} & \text{para } x = 1, 2, \ldots, N-1 \end{cases} \tag{A.3}$$

[1] De forma muy cercana a la de Karhunen Loéve.

Las correspondientes versiones de la transformada bidimensional se definen como (A.4) y (A.5).

$$F(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) cos \frac{(2x+1)u\pi}{2M} cos \frac{(2y+1)v\pi}{2N} \tag{A.4}$$

$$f(x,y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \alpha(u)\alpha(v) F(u,v) cos \frac{(2x+1)u\pi}{2M} cos \frac{(2y+1)v\pi}{2N} \tag{A.5}$$

Donde igualmente debemos cumplir la Ecuación (A.3).

El núcleo de la transformada discreta del coseno para N=4 y N=8 se puede puede ver gráficamente en la Figura A.1.

Algunas de las principales propiedades de la DCT son:

- Es ortogonal y separable.
- La DCT es una matriz real (lo que indica que no es completamente reversible dado que opera con números en coma flotante).
- Significado físico muy similar a DFT.
- No aparece replicada y no tiene simetrías.
- No posee la propiedad de convolución.
- La DCT concentra mucho la información en los primeros coeficientes (gran índice de compactación espectral), es la más usada en codificación y compresión (dado que existen algoritmos rápidos basados en técnicas matriciales y en la FFT).

Además como curiosidad comentar que es la transformada utilizada en el proceso de compresión JPEG.

El porqué de utilizar la DCT y no la FFT, por ejemplo, es debido a que la DCT tiene un mayor índice de compactación espectral, lo que provoca que pueda aproximar señales lineales con menor número de componentes. En la Figura A.2 se puede apreciar que tras realizar la transformación, truncar algunos resultados y volver a aplicar la inversa correspondiente a cada una de las funciones el resultado obtenido por la DCT es muy superior al de la FFT.

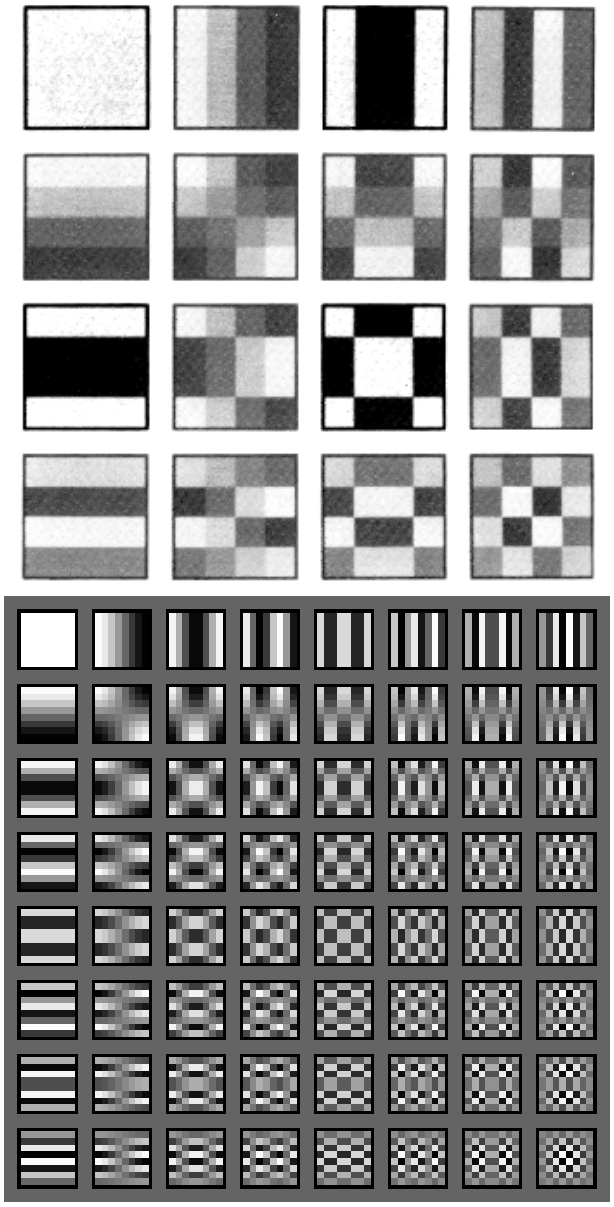


Figura A.1: Núcleo de la DCT para N=4 y para N=8 respectivamente.

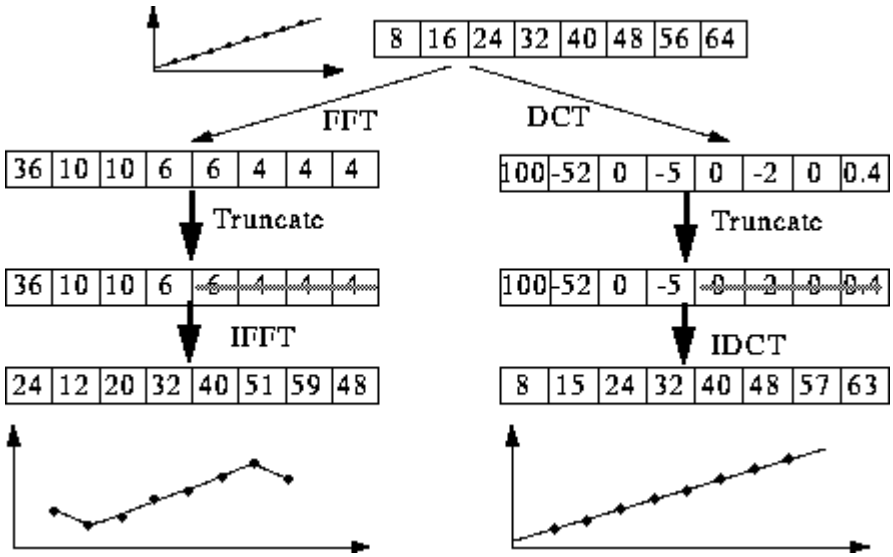


Figura A.2: FFT vs. DCT

Apéndice B

Espacios de Color

Una imagen se puede representar utilizando distintos espacios de color.

Uno de los más conocidos es el espacio RGB24, donde se utilizan un total de 24 bits dispuestos en tres grupos de 1 Byte. Cada uno de ellos representa una de las componentes de color del pixel (rojo, verde y azul).

También podemos encontrar el RGB32, donde se añade un último Byte que representa el canal Alfa.

Pero no sólo podemos representar la información de cada uno de los pixels mediante colores, también podemos hacerlo mediante la luminancia (Y) y la crominancia (C_b y C_r). Como el sistema visual humano percibe con más facilidad la pérdida de información en la luma que en la croma las bandas C_b y C_r se suelen muestrear [12].

Algunos ejemplos de este submuestreo los podemos ver en la Figura B.1 y son los siguiente:

- **Formato 4:4:4**: Se muestren las tres componentes con la misma frecuencia, es decir, tenemos información única de luminancia y crominancia para cada pixel.
- **Formato 4:2:2**: Cada dos muestras de Y se toma una muestra de C_b y otra de C_r (sólo en la horizontal).
- **Formato 4:1:1**: Cada cuatro muestras de Y se toma una muestra de C_b y otra de C_r (igualmente, sólo en la horizontal).
- **Formato 4:2:0**: Cada dos muestras de Y en horizontal y en vertical se toma una de C_b y otra de C_r, pero en los puntos intermedios.

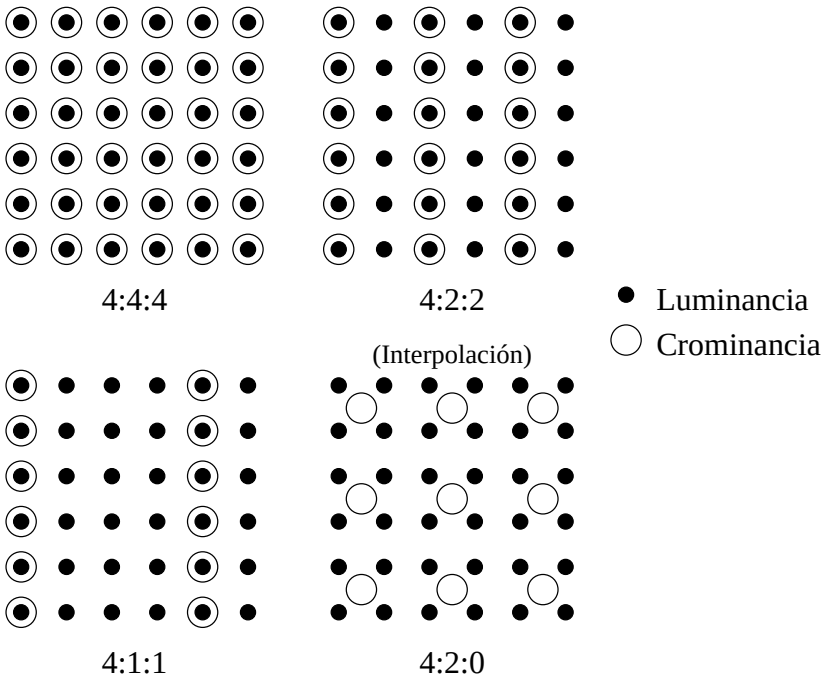


Figura B.1: Submuestreo de la crominancia en los distintos espacios YUV

Apéndice C

Material Informático Utilizado

Para la realización de este proyecto se ha utilizado el siguiente material:

- Sistema Operativo `GNU/Linux` con distintas versiones del kernel (2.4.20, 2.6.0 y 2.6.6).
- Compilador `GNU/GCC` [13] en su versión 2.95.
- Debugger `GNU/GDB` [14] para la depuración del código, versión 2002-04-01-cvs.
- Biblioteca gráfica `SDL` [11], versión 1.2.7.
- Una modificación de la biblioteca multimedia `FFMPEG` [10]. La versión sobre la que se hicieron las modificaciones fué la 0.4.7.
- Biblioteca `Avifile` [9], versión 0.7.
- Editor `XEmacs` [15] como entorno de programación y de edición LaTeX, versión 21.4.15.
- Planificador de compilaciones LaTeX `Rubber` [16], versión 0.99.6.
- Equipo Intel Pentium 4 con una frecuencia de reloj de 1.7GHz, 256MB de memoria RAM y microprocesador gráfico de 64 MB de memoria DDR. Con posibilidad de ejecutar instrucciones MMX y SSE2.
- Equipo Intel Pentium III con una frecuencia de reloj de 450MHz, 128MB de memoria RAM y microprocesador gráfico de 32 MB de memoria. Con posibilidad de ejecutar instrucciones MMX y SSE.

- Equipos Intel Celeron PIII con una frecuencia de reloj de 800Mhz, 128MB de memoria RAM y microprocesador gráfico integrado en la placa base sin aceleración 3D. Con posibilidad de ejecutar instrucciones MMX y SSE.

Bibliografía

[1] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[2] Douglas E. Comer. *Internetworking with TCP/IP. Principles, Protocols,and Architectures*, volume I. Prentice Hall, 4th edition, 2000.

[3] Juan Mariano de Goyeneche. *Multicast over TCP/IP HOWTO*. Available from World Wide Web: http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/Multicast-HOWTO.html.

[4] Ken Lindahl. IP Multicast - LAN to WAN. Technical report, UC Berkeley, October 2002.

[5] Rob Koenen. Overview of the MPEG-4 Standard. Technical report, International Organisation for Standardisation, March 2002.

[6] Steven Gringeri, Roman Egorov, Khaled Shuaib, Arianne Lewis, and Bert Basch. Robust Compression and Transmission of MPEG-4 Video. Technical report, GTE Laboratories Incorporated, 2003. Available from World Wide Web: http://www.kom.e-technik.tu-darmstadt.de/acmmm99/ep/gringeri.

[7] Touradj Ebrahimi and Caspar Horne. MPEG-4 Natural Video Coding - An overview. Technical report, Swiss Federal Institute of Technology, 2003. Available from World Wide Web: http://3c.nii.org.tw/3c/silicon/embedded/MPEG/MPEG.htm.

[8] Fernando Pereira and Touradj Ebrahimi. *The MPEG-4 Book*. Prentice Hall PTR, 2002.

[9] Linux AVI file library. Available from World Wide Web: http://avifile.sourceforge.net/.

[10] FFMPEG Multimedia System. Available from World Wide Web: http://ffmpeg.sourceforge.net.

[11] Simple DirectMedia Layer. Available from World Wide Web: http://www.libsdl.org/.

[12] Vicente González Ruiz. Apuntes de Imagen y Sonido. Capítulo 8: JPEG, 2004.

[13] GNU/GCC Home Page. Available from World Wide Web: http://gcc.gnu.org/.

[14] GNU/GDB Home Page. Available from World Wide Web: http://www.gnu.org/software/gdb/gdb.html.

[15] XEmacs Home Page. Available from World Wide Web: http://www.xemacs.org/.

[16] Rubber Home Page. Available from World Wide Web: http://rubber.sourceforge.net/.