PROYECTO FIN DE MÁSTER

POSGRADO EN INFORMÁTICA

UNIVERSIDAD DE ALMERIA

ESCUELA POLITÉCNICA SUPERIOR

MÁSTER EN INFORMÁTICA INDUSTRIAL

IMPLEMENTACIÓN DE UN SERVIDOR DE VÍDEO ESCALABLE EN EL TIEMPO

Curso 2011/2012

Alumno/a:
Aquilino Gallardo Maldonado

Director/es:
Vicente González Ruiz

TRABAJO FIN DE MÁSTER

MÁSTER EN INFORMÁTICA INDUSTRIAL

POSGRADO EN INFORMÁTICA

IMPLEMENTACIÓN DE UN SERVIDOR DE VÍDEO ESCALABLE EN EL TIEMPO

Autor:

Aquilino Gallardo Maldonado

Director:

Vicente González Ruiz

Almería, Junio de 2012

Este proyecto está dedicado

a mi familia y amigos,

que siempre confiaron en mí.

Agradecimientos

Mi más sincero agradecimiento a mis compañeros de clase por su apoyo y amistad, en especial a César Pardo que sin su ayuda no se podría haber llevado a cabo este proyecto.

A mi director de proyecto, Vicente González Ruiz, por su entrega, compromiso y dedicación en todos y cada uno de los puntos de este proyecto.

A todos ellos, GRACIAS.

Paul Leary: Es genial trabajar con ordenadores. No discuten, lo recuerdan todo y no se beben tu cerveza.

Marvin Minsky: Es ridículo vivir 100 años y sólo ser capaces de recordar 30 millones de bytes. O sea, menos que un compact disc. La condición humana se hace más obsoleta cada minuto.

ÍNDICE

Una arquitectura cliente/servidor para el streaming de vídeo escalable en el tiempo

Aquilino Gallardo Maldonado
Vicente González Ruiz*

***Resumen*—En este proyecto se ha implementado la parte servidora de un servicio de streaming de vídeo escalable en el tiempo. El vídeo a enviar se ha comprimido previamente mediante MCTF y la tarea del servidor consiste en enviar dicho vídeo a un conjunto de clientes. Lo novedoso de este proyecto es que tanto el servidor como el cliente se comunican, para saber qué cantidad de vídeo puede transmitirse en cada instante, en función del ancho de banda disponible, sin que esto provoque interrupciones en su reproducción. Finalmente, mediante una serie de pruebas, se ha comprobado el correcto funcionamiento del protocolo y de la gestión de las diferentes capas de vídeo por parte del servidor.**

Palabras clave—MCTF, escalabilidad temporal, gestión de capas, tasa de transmisión.

Abstract—This project has implemented the server side of a video streaming service scalable in time. The video to send, it has been previously compressed by MCTF and server task is to send the video to a set of clients. The novelty of this project is that both the server and client communicate, to know how much video can be transmitted at any instant, depending on available bandwidth, without this leading to interruptions in playback. Finally, through a series of tests, we have found the correct protocol operation and management of the different layers of video from the server.

Keywords— MCTF, temporal scalability, layer management, transmission rate.

I. Introducción

DESDE la invención de la red Internet a finales de los años sesenta, los diseñadores y usuarios de la red han intentado utilizarla para los más diversos propósitos. Inicialmente, sólo aplicaciones como la transmisión de ficheros, el correo electrónico o el chat estaban disponibles. Como se puede comprobar, incluso desde estos tiempos tan iniciales ya podían distinguirse dos tipos de aplicaciones: (1) aquellas, como el correo electrónico, en las que las restricciones temporales impuestas por la propia aplicación demandan un uso del ancho de banda muy bajo y flexible, y otras (2), como son la transmisión del ficheros o el chat, donde o es imprescindible disfrutar de una cantidad suficiente de ancho de banda para transmitir dichos ficheros rápidamente o además, como ocurre con el chat, el requerimiento más importante radica en que la latencia de la red debe ser lo más pequeña posible.

Para acomodar a estos dos tipos de tráfico existen dos filosofías básicas: (1) distinguir entre ambos tipos de flujos de datos (streams), tratándoles de forma diferente (fundamentalmente, dedicando mucho más ancho de banda aunque sea de forma puntual a las aplicaciones de la segunda clase) y (2), no diferenciarlas, y esperar que con la evolución de la propia red (que prácticamente ha doblado su tamaño y capacidad cada año desde su nacimiento) el servicio ofrecido sea suficientemente bueno. Por este motivo se han creado un número considerable de tecnologías de transmisión, algunas especialmente concebidas para transmitir flujos de datos multimedia como puede ser ATM (Asynchronous Transfer Mode), y otras más pensadas para transmitir datos en general, de las cuales sin duda la más exitosa a sido Ethernet.

Actualmente ambos tipos de tecnologías coexisten; las especializadas en flujos multimedia y las no especializadas, aunque en redes para usos diferentes debido fundamentalmente a motivos económicos. El resultado es que, para transmitir datos (y hoy en día en la Internet pública no se realiza un tratamiento diferente de los datos en función de su contenido) no se puede conocer qué cantidad de ancho de banda vamos a poder utilizar cuando transmitimos. Cuando enviamos un correo electrónico esto va a provocar que el tiempo que tarda el correo en llegar hasta el receptor pueda variar unos pocos segundos. Sin embargo, cuando enviamos datos sensibles al tiempo, como puede ser una conversación telefónica o un vídeo, podría ocurrir que el ancho de banda ofrecido en ese momento por la red fuera insuficiente. En este último caso, la única solución válida a este problema consiste en precargar una cantidad de vídeo concreta en los receptores antes de comenzar la reproducción. Dicho tiempo de precarga lo calcula el receptor que, en función del bit-rate del vídeo, su duración y del bit-rate de la red estima la cantidad de datos que deben precargarse para que, en el caso de que ambos bit-rates se mantengan más o menos durante todo el tiempo que dura el vídeo, este pueda reproducirse sin pausa.

En este trabajo se presenta una solución a la transmisión de vídeo bajo demanda (en inglés, Video on Demand o VoD) sobre la red Internet. Para ello se plantea una arquitectura cliente/servidor y se transmite un tipo de vídeo comprimido especialmente diseñado para minimizar la precarga de datos por parte del usuario. Se estudian la posibilidades de un sistema VoD que utiliza un codificador de vídeo escalable que permite maximizar la calidad del vídeo reproducido (en concreto, el número de imágenes por segundo de la reproducción) en función de bit-rate que existe en ese momento disponible para el usuario en la red. El resultado es un sistema de visualización remota de vídeos con una muy baja latencia inicial, ya que apenas existe precarga, y una alta resistencia a los cortes de la reproducción.

*Departamento de Arquitectura de Computadores y Electrónica de la Universidad de Almería.

El resto del documento se organiza de la siguiente manera. En la Sección II se describe el funcionamiento básico de los sistemas de streaming de vídeo bajo demanda. La Sección III está dedicada a mostrar los fundamentos del codificador de vídeo escalable usado. La Sección IV se analizan las printipales propiedades de los sistemas cliente/servidor. En la Sección V se presenta la solución aportada en este trabajo fin de máster. En la Sección VI se realiza una evaluación de dicha propuesta. Finalmente, la Sección VII expone las principales conclusiones a las que se han llegado y líneas de trabajo futuro más interesantes.

II. Los sistemas VoD

En un sistema VoD el usuario selecciona un vídeo (generalmente previamente almacenado y comprimido, evidentemente, junto con su audio asociado) que está almacenado en un servidor remoto, y lo visualiza (previa transmisión) de la misma manera que si estuviera almacenado en su reproductor de vídeo local, esto es, iniciando, deteniendo, rebobinando hacia delante y o hacia detrás, y pudiendo acceder a cualquier instante de tiempo del vídeo cuando lo desea. Generalmente los tiempos de retardo en cada una de estas acciones son de unos pocos segundos.

Los primeros sistemas VoD aparecieron a principios de los años noventa y estaban implementados sobre redes dedicadas, especialmente sobre ATM. Hoy en día existen multitud de servidores VoD, algunos para contenidos de pago (como puede ser MegaVideo) y otros con contenidos de acceso libre. Entre estos últimos, el más famoso sin duda alguna es YouTube.

Cuando un usuario utiliza su navegador Web (parte cliente) para visualizar un vídeo almacenado en YouTube (parte servidora), el cliente realiza una petición al servidor que, mediante HTTP (HyperText Transfer Protocol), finaliza en la transmisión interactiva de una secuencia de vídeo comprimido.

Aunque el servidor probablemente ofrezca diferentes versiones del mismo vídeo, con distintas resoluciones espaciales, el navegador precarga una cierta cantidad de datos antes de comenzar la reproducción con el objetivo anteriormente mencionado: reducir o eliminar las pausar provocadas por la falta de ancho de banda. Si dicha precarga es insuficiente, el usuario tiene solamente una opción: detener la reproducción actual e intentar de nuevo con una versión de menor resolución (que seguramente tendrá un bit-rate de codificación menor). Como es evidente, dicha técnica es sólo una solución parcial ya que el número de resoluciones no suele ser superior a dos o tres, con lo que no existe en realidad un número alto de posibilidades para acomodar el bit-rate de codificación al bit-rate de la red (ancho de banda).

En este trabajo se plantea el uso de un compresor de vídeo que ofrece muchas más posibilidades a la hora de encajar ambos bit-rates. Además, la conmutación entre dichas posibilidades se realiza de forma transparente para el usuario. En concreto, el usuario recibe de un vídeo cuya resolución temporal (el número de imágenes por segundo) depende de la capacidad instantánea de canal de transmisión. Cuando el número de imágenes/segundo se reduce, el reproductor interpola las imágenes que faltan para que la sensación de movimiento percibida no se vea afectada.

III. La codificación de vídeo escalable

Describiremos el sistema de codificación de vídeo escalable a partir de sucesivas mejoras de codecs de vídeo estándar.

III-A. JPEG

El estándar de compresión de imágenes JPEG [1] (Joint Photographic Experts Group) fué desarrollado por la ISO en 1992 [2] y es el compresor de imágenes lossy por excelencia. No en vano está soportado por casi todos los dispositivos de tratamiendo multimedia. El gran motivo de su éxito radica en que el codec JPEG está especialmente diseñado para comprimir imágenes a color a una tasa de 1 bits/pixel (bpp) sin que un humano sea capaz normalmente de diferenciar la imagen comprimida de la original.

EL algoritmo de compresión consiste básicamente en lo siguiente:

1. Convertir la imagen al dominio YCbCr.
2. Submuestrear la crominancia.
3. Desplazar cada componente al rango $[-128, 127]$.
4. Para cada componente (Y, Cb y Cr):
   a) Aplicar la (8×8)-DCT a cada componente.
   b) Cuantificar los coeficientes DCT.
   c) Codificar entrópicamente los coeficientes cuantificados.

A continuación iremos describiendo cada una de estas fases con mayor detalle:

III-A1. RGB → YCbCr: Consiste en cambiar del dominio de color RGB al YCbCr. Esto se realiza mediante la transformada [3]

$$\begin{pmatrix} \text{Y} \\ \text{Cb} \\ \text{Cr} \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,144 \\ -0,1687 & -0,3313 & 0,5 \\ 0,5 & -0,4187 & -0,0813 \end{pmatrix} \begin{pmatrix} \text{R} \\ \text{G} \\ \text{B} \end{pmatrix}. \tag{1}$$

III-A2. Submuestreo de la crominancia: El sistema visual humano es más sensible a la luminancia (Y) que a la crominancia (CbCr). Por tanto, la información sobre el color de una imagen puede submuestrearse sin que sea perceptible. El patrón de submuestreo más frecuentemente usado es el 4:2:2 (véase la Figura 1).

III-A3. $[0, 255] \rightarrow [-128, 127]$*:* Las muestras se desplazan al rango $[-128, 127]$ con el objetivo de que la media sea 0. Esto es necesario para que el rango dinámico de la DCT (Discrete Cosine Transform) sea mínimo.

III-A4. 8x8 2D-DCT: Las muestras $I[n]$ son trasformadas usando la DCT que responde a:

$$\text{DCT}[u] = \frac{\sqrt{2}}{\sqrt{N}} K(u) \sum_{n=0}^{N-1} I[n] \cos \frac{(2n+1)\pi u}{2n}, \tag{2}$$

donde

$$K(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{si } u = 0 \\ 1 & \text{if } u > 0. \end{cases} \tag{3}$$

La DCT es separable lo que significa que la 2D-DCT se calcula aplicando la DCT primero a las filas de la imagen y luego a las columnas (o viceversa).

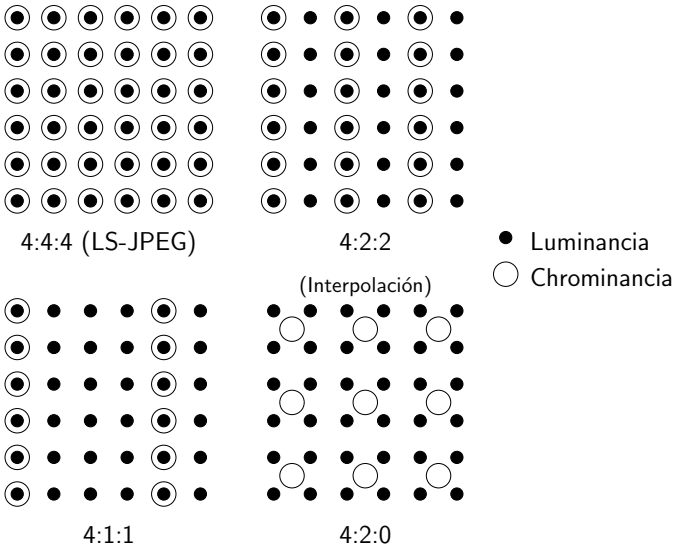

Figura 1. Patrones de sumuestreo de la crominancia usado en los estándares ISO.

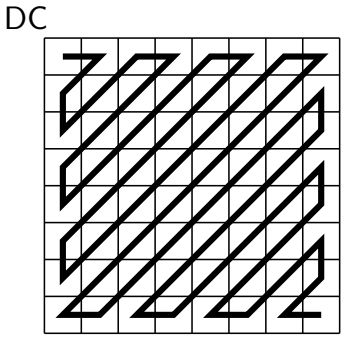

Figura 3. Recorrido en zig-zag de los coeficientes AC en JPEG.

III-A5. Cuantificación: Los coeficientes DCT se cuantifican con el objeto de eliminar fundamentalmente el ruido de la imagen. Esta fase es similar a un proceso de filtrado en que se eliminan las altas frecuencias de la imagen.

El JPEG propone usar las matrices de cuantificación que se muestran en la Figura 2. Este proceso queda descrito por la ecuación

$$\text{2D-DCT}'[u,v] = \text{round}\Big(\frac{8\times 8\text{-DCT}[u,v]}{\text{Z}[u,v]}\Big), \qquad (4)$$

donde $\text{Z}[\cdot,\cdot]$ es la correspondiente matriz de cuantificación.

III-A6. Codificación entrópica: Finalmente los coeficientes DCT cuantificados se comprimen usando un código de Huffman. Más concretamente:

1. Eliminar al el coeficiente DC de la loseta actual el coeficiente DC de la anterior loseta siguiendo un recorrido de tipo *raster*. Esto elimina la correlación entre bloques.
2. Los coeficientes AC de recorren en *zig-zag* (véase la Figura 3). Este recorrido responde a un incremento en las componentes de frecuencia.
3. Los runs generados por el recorrido anterior se comprimen usando un código de longitud variable que asigna a aquellos runs más frecuentes códigos de longitud menor.

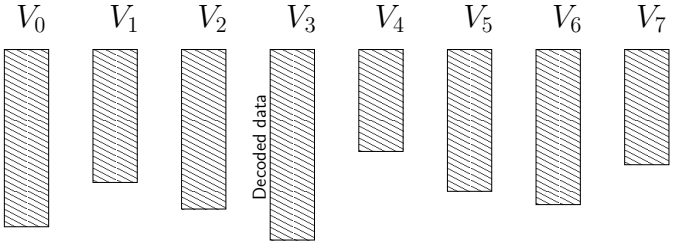

Figura 4. Un ejemplo de las longitudes asociadas a cada imagen en MJPEG.

III-B. Motion JPEG (MJPEG)

El estándar de compresión de vídeo MJPEG [4] es básicamente una aplicación del estándar de compresión de imágenes JPEG [5].

Un aspecto importante del codec MJPEG es que a calidad constante el tamaño de cada imagen puede ser diferente dependiendo del contenido visual de las mismas (véase la Figura 4.

III-C. MPEG-1

III-C1. Overview del codec: El estándard MPEG-1 [6] (formalmente llamado *Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s* o *Estándar ISO/IEC 11172*) define cómo debe descomprimirse un bit-stream MPEG-1, consiguiendo una tasa de bits de aproximadamente 1.5 Mbit/s (lo que se corresponde con uno ratio de compresión de aproximadamente 180:1), para una señal de 25 imágenes por segundo con una profundidad de color de 24 bits y un tamaño por imagen de 640x480 puntos.

Un compresor MPEG-1 es un sistema híbrido que combina un codec lossy-DPCM aplicado a las imágenes de la secuencia de vídeo y un compresor de imágenes semejante al usado en JPEG. Si observamos la Figura 5 podemos ver que se trata básicamente de un codec lossy-DPCM de orden 1 y que elimina la redundancia temporal mediante estimación de movimiento, en inglés, Motion Estimation (ME). Como puede apreciarse, para cada imagen de entrada V_i se genera una imagen residuo $E_i^{[q]}$, cuya compresión entrópica usando codificación de longitud variable, en inglés Variable Length Coding (VLC) genera, junto con la información sobre el movimiento detectado en el vídeo por el estimador de movimiento el codestream MPEG-1.

Por otra parte, tal y como puede apreciarse en la parte inferior de la Figura 5, el descompresor utiliza dicho residuo junto con la imagen previamente reconstruida para recuperar una versión aproximada $V_i^{[q]}$ de la secuencia original. En todas estas expresiones, el superíndice $[q]$ indica el nivel de cuantificación utilizado por el cuantificador $\mathcal{Q}^q$. Dicha cuantificación reduce la precisión numérica de los coeficientes de la transformada espacial $\mathcal{T}$ que en este caso se trata de la DCT (Discrete Cosine Transform, aplicada en losetas de 8x8 píxeles) de las imágenes residuo E_i resultantes de restar a la imagen actual la imagen predicción generada por la imagen anterior que ha sido reconstruida por el descompresor. En otras palabras,

$$E_i = V_i - \mathcal{P}(V_{i-1}^{[q]}) \qquad (5)$$

Luminance

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Chrominance

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Figura 2. Matrices de cuantificación propuestas por el JPEG.

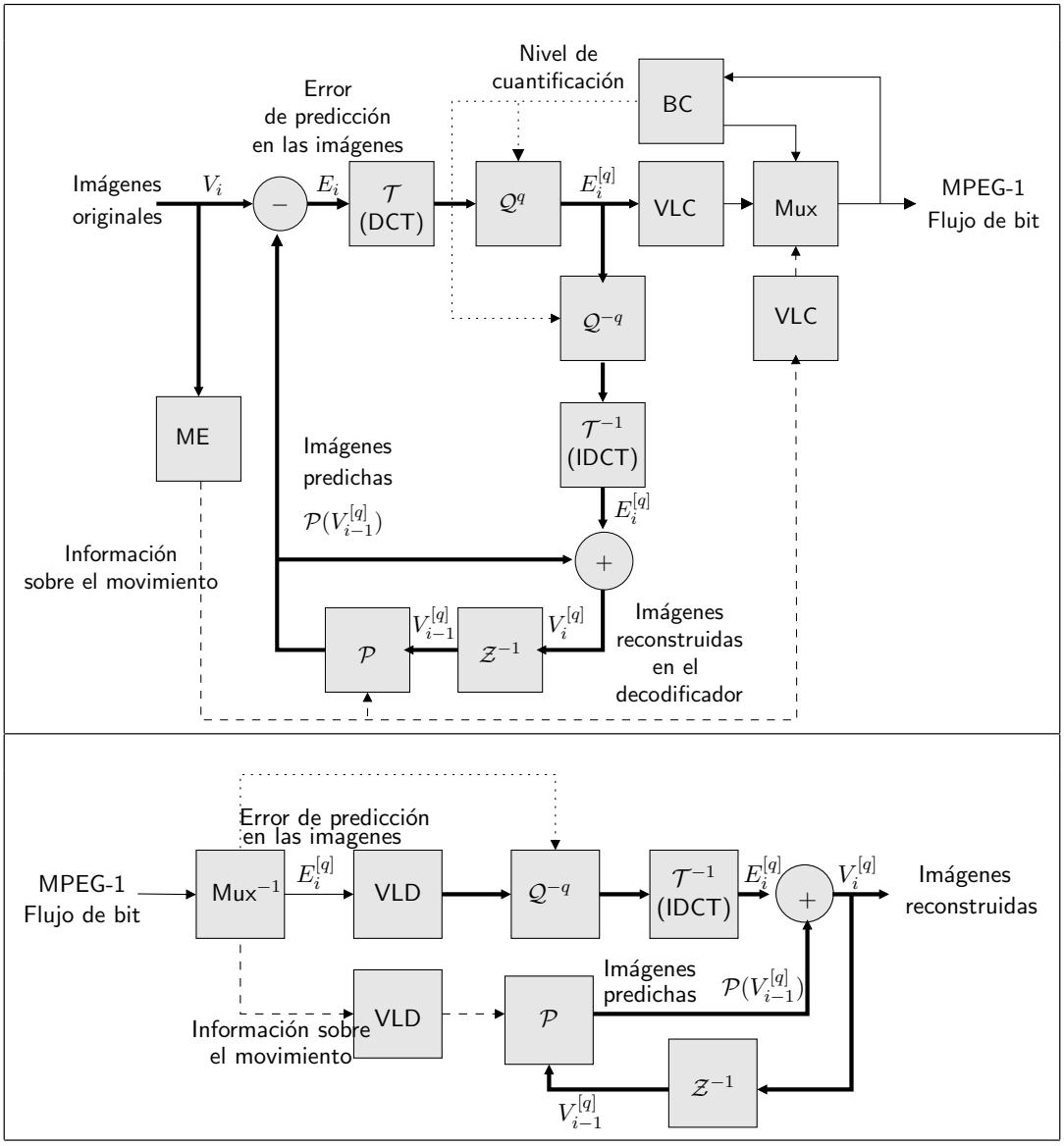

Figura 5. El codec MPEG-1. Arriba el compresor y debajo el descompresor.

donde $\mathcal{P}(V_{i-1}^{[q]})$ es la predicción usando estimación de movimiento para generar la imagen V_i a partir de la imagen $V_{i-1}^{[q]}$, que es la última imagen generada por el descompresor. Las imágenes

$$V_i^{[q]} = \mathcal{T}^{-1}\big(\mathcal{Q}^{-q}(E_i^{[q]})\big) + \mathcal{P}(V_{i-1}^{[q]}) \tag{6}$$

donde

$$E_i^{[q]} = \mathcal{Q}^q(E_i). \tag{7}$$

Tal y como se ha descrito, MPEG-1 es un compresor de vídeo lossy debido a la etapa de cuantificación $\mathcal{Q}$, que aplica un nivel de cuantificación generalmente variable y que dependiente del bit-rate instantáneo producido. Dicho nivel de cuantificación es controlado por el módulo BC (Bit-rate Control) y la información resultante de dicha estimación del nivel de cuantificación se incluye, junto con el residuo de las texturas cuantificadas ($E_i^{[q]}$) y la información sobre el movimiento, en el bit-stream que el compresor produce y consume el descompresor.

III-C2. Estimación de movimiento (ME): Consiste en la eliminación de la redundancia temporal, es decir, la información que se repite en una secuencia de imágenes. Se trata de localizar regiones en las imágenes que se mantengan invariables en posteriores imágenes de la secuencia de vídeo, de manera que sólo es necesario almacenar la diferencia entre dichas imágenes, ya que el resto es igual y no es necesario almacenarlo repetidamente. Esta diferencia entre la imagen anterior y posterior o posteriores en la secuencia, es codificada en lo que se denomina una imagen de predicción. Es evidente, que el proceso de compresión en la estimación de movimiento se lleva a cabo una vez para cada una de esas regiones invariables, sin embargo, la descompresión se hará tantas veces como dicha región aparezca en las imágenes posteriores de la secuencia.

Dependiendo de la manera en la que son construidas estas predicciones, en MPEG-1 se encuentran los siguientes tipos de codificación de imágenes:

1. **Intra-coded:** Una imagen intra (I) es aquella en la que su codificación depende únicamente de la propia imagen y de ninguna otra. Esto es, no hay necesidad de llevar a cabo una estimación del movimiento, y por tanto, no contiene información sobre vectores de movimiento.
2. **Predictive-coded:** Las imágenes denominadas P, son aquellas cuya codificación depende de *una* imagen previa (en la secuencia de imágenes). Esta imagen de la que depende P puede ser otra imagen P o una imagen I.
3. **Bidirectionally predictive-coded:** Al igual que una imagen P, la dependencia de las B puede depender de imágenes I o P, con la salvedad de que una imagen B depende de *dos* imágenes, no sólo de una. Por tanto, la dependencia es bidireccional, es decir, depende de una imagen anterior y otra posterior, tomando en cuenta la secuencia de imágenes.

En la Figura 6 se exponen las distintas dependencias entre los tres tipos de imágenes, que se generan al codificar una imagen a partir de otra u otras. En dicha figura se representan las imágenes tipo I, P y B y sus dependencias en los colores azul, verde y rojo, respectivamente.

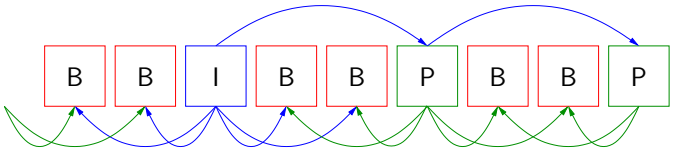

Figura 6. Dependencia de imágenes I, B y P.

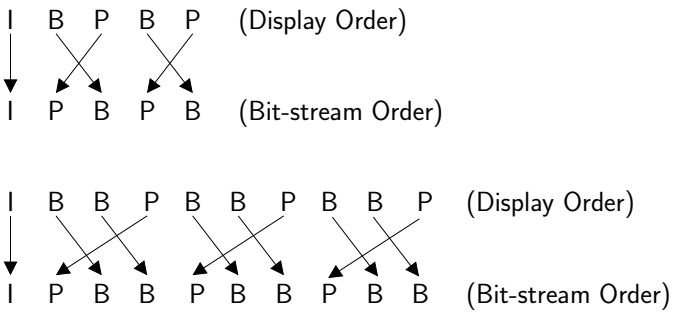

Figura 7. Ejemplos de ordenación de imágenes en un GOP.

En todos los compresores de vídeo existe el concepto de GOP (Group Of Pictures) que no es más que la forma en que las imágenes del vídeo se descorrelacionan. En la mayoría de los GOP's la primera imagen es intra-codificada y el resto, cada una depende de la anterior (P) o de dos imágenes, una anterior y otra posterior (B). En la Figura 6 se muestra un ejemplo de GOP BBIBBPBBP, lo que significa que el resto del vídeo está codificado siguiendo un patrón que se forma repitiendo dicho GOP tantas veces como sea necesario. En la Figura 7 se presentan otros ejemplos diferentes de GOP.

Como puede deducirse de este patrón de descorrelación temporal, para la descompresión de una imagen $V_i^{[q]}$, puede ser necesario, por ejemplo, descomprimir la imagen previa, la $V_{i-1}^{[q]}$, y así sucesivamente hasta que se alcance una imagen I. Evidéntemente, esto no es lo ideal cuando, por ejemplo, queremos descomprimir única y directamente la imagen $V_i^{[q]}$. Además, si una imagen con la que existe una dependencia temporal ha sido perdida por algún error de almacenamiento o transmisión, las imágenes que dependen de ella carecen de toda la información necesaria para ser descomprimidas y, por tanto, poder ser visualizadas. Siendo conscientes de esta limitación que impone el sistema, en necesidad de un mayor número de dependencias conforme mayor ratio de compresión se desee, es evidente que es necesario adoptar un compromiso entre ambos.

En este contexto, las imágenes son comprimidas como un todo, denominando a esta unidad GOP. Nótese que el concepto de GOP funciona del mismo modo usando imágenes P, como imágenes B. Cuando son frames (imágenes de la secuencia) tipo B en un GOP, podemos hablar sobre diferentes tipos de ordenación de las imágenes (en la Figura 7 se presentan algunos ejemplos):

1. **Display order:** Es el orden usado para mostrar las imágenes, es decir, el orden en que la secuencia de imágenes forma el vídeo al sucederse.
2. **Bit-stream order:** Es el orden en el cual las imágenes son almacenadas.

A	X_1	B	$X_1 = (A + B)/2$
X_2	X_3	X_4	$X_2 = (A + C)/2$
C	X_5	D	$X_3 = (A + B + C + D)/4$

Figura 8. Interpolación de píxeles en MPEG1.

Por otra parte, en relación a la forma en que se diseñan los GOPs, en una secuencia MPEG-1 se pueden encontrar dos tipos de GOP:

1. **Cerrado:** Un GOP cerrado no necesita de otros GOP para ser descomprimido. Necesariamente, es aquel en el que su primera imagen es tipo intra.
2. **Abierto:** Por otra parte, un GOP abierto indica que él mismo no tiene toda la información para descomprimir las imágenes que contiene, dependiendo de este modo, de otros. Los GOP abiertos son muy poco usuales puesto que no permiten un acceso aleatorio a las imágenes de la secuencia, ni soportan mecanismos de control de errores.

Las referencias a otras imágenes están especificadas a porciones de la imagen cuadradas que se denominan macro-bloques. Un macro-bloque es una región de una imagen de 16x16 píxeles. Cada (macro-)bloque se busca en la imagen que se quiere predecir y las mejores coincidencias se usan para construir un vector de movimiento, el cual indica donde se encontrara esa región de la imagen en el siguiente instante de tiempo. Dependiendo de si la búsqueda en bloques coincidentes en ambas imágenes a sido exitosa o no y al número de éstas, es decir, el número de referencias encontradas, los macro-bloques son clasificados en:

1. **I:** Cuando la compresión de los bloques residuo genera mas bits que el original. Por tanto, no se suele comprimir.
2. **P:** Cuando es mejor comprimir el bloque residuo y hay sólo una referencia a un macro-bloque.
3. **B:** Idem a P, pero hay dos referencias a macro-bloques.
4. **S (Omitidos):** Cuando la energía del bloque residuo es menor que la dada por el umbral establecido. Por tanto, al no ser relevante para el conjunto de la imagen, no se codifica nada.

La estimación del movimiento puede ser llevada a cabo usando la resolución real de la imagen, o identificando una localización intermedia entre varios píxeles como su fracción. En la Figura 8 se ilustra dicho concepto.

Ahora bien, ¿qué criterio usa el compresor para determinar si efectivamente dos macro-bloques presentan una referencia entre ellos, siendo útil como predicción de movimiento? Existen varios como son la diferencia de error o error entrópico. El mas usado se denomina criterio de similitud y su forma de funcionar consiste en medir la diferencia o error entre dos macro-bloques. Ésta medición se lleva a cabo de dos formas: Los macro-bloques que se quieren comparar están representados por a y b.

1. Medida del error al cuadrado

$$\frac{1}{16 \times 16} \sum_{i=1}^{16} \sum_{j=1}^{16} (a_{ij} - b_{ij})^2$$

2. Medida del error absoluto

$$\frac{1}{16 \times 16} \sum_{i=1}^{16} \sum_{j=1}^{16} |a_{ij} - b_{ij}|$$

Pero definamos primero cómo se llevan a cabo básicamente estas búsquedas. Hay dos parámetros básicos en toda búsqueda, y son: qué buscar y dónde buscarlo. Lo que se busca es un macro-bloque que esté en 2 imágenes consecutivas de la secuencia, llamadas imagen de referencia e imagen predicha. Como ya hemos visto, determinar si esta o no, depende de los "criterios de coincidencia" que usemos, ya que no sólo se buscan macro-bloques idénticos, sino con un parecido razonable. Lo que ocurre es que buscar en toda la imagen un macro-bloque es muy costoso computacionalmente y además innecesario. Las búsquedas mas productivas se llevan a cabo considerando que el elemento que enmarca el macro-bloque tiene una limitación de desplazamiento de un frame de la secuencia a otro, y por tanto, es muchísimo mas productivo buscar un macro-bloque en las inmediaciones de donde estaba en el frame anterior. Esta región de la imagen se denomina "área de búsqueda". A continuación se exponen los distintos tipos de búsqueda:

1. **Búsqueda completa:** Haciendo un uso intensivo de la CPU, se comprueban todas las posibilidades. La ventaja es que se encuentran todas las coincidencias existentes y, por tanto se llega a la mejor compresión posible para MPEG1.
2. **Búsqueda logarítmica:** Es una versión de la 'búsqueda completa', pero de mayor eficiencia. Esto se consigue al reducir los macro-bloques y las áreas de búsqueda, al trabajar sobre un nivel de resolución de la imagen menor al 1:1, tanto como se desee. Una vez encontrada la mejor coincidencia entre macro-bloques, se transforma[1] la imagen al siguiente nivel de resolución, acercándonos a las proporción 1:1. Y de igual modo se adaptan la información recogida (las referencias o vectores de movimiento) hasta el momento a la nueva resolución, esto es, aumentando también las proporciones del propio macro-bloque y reubicando la referencia al mismo, en un área de tan sólo ± 1. Es decir, se lleva a cabo una búsqueda completa, pero en una imagen mas pequeña, esto es así porque la hemos reducido, cuantas veces queramos, a un nivel resolución X veces menor al original [7], [8]. De esta manera obtenemos dos ventajas: La primera y evidente es que al aplicar la búsqueda en una imagen mas pequeña, el proceso se lleva a cabo requiriendo mucho menos tiempo. La segunda es que la granularidad de los vectores de movimiento encontrados se progresiva y de alta calidad.

[1]La transformación consiste en aplicar la transformada Wavelet e inversa de Wavelet para llevar la imagen a niveles resolución menores o mayores, respectivamente.

En primer lugar, el algoritmo es progresivo. Al tener la imagen en una resolución menor, los datos que conserva dicha imagen son los pertenecientes a los elementos mas grandes y notorios, descartándose de esta forma tan sencilla todos los pequeños detalles de la imagen. De esta manera los primeros vectores de movimiento que se encuentran son los mas importantes, ya que ellos predicen el movimiento de los elementos más visibles en la imagen. Así tenemos un sistema que proporciona las referencias más importantes desde el primer instante de ejecución y según le demos tiempo, nos provee no sólo de mas referencias, sino también de una mayor precisión en las referencias encontradas anteriormente.
Y en segundo lugar, proporciona estimaciones de calidad. Esto es necesario porque para que la predicción del movimiento sea satisfactoria desde un punto de vista subjetivo del espectador, ésta debe ser precisa, de forma que no se solapen distintos elementos de la imagen. En esta línea esta demostrado que el mejor método de precisar el origen y destino de un macro-bloque en una secuencia de imágenes consiste precisamente en determinar el elemento al que pertenece dicho macro-bloque y después ir precisando su ubicación a nivel de píxeles, y no directamente esto último. Como vemos, de nuevo este proceso viene dado de forma inherente al empezar a trabajar con las imágenes en niveles de resolución inferiores. Aquí nos podemos percatar de la gran utilidad que tiene el trabajar sobre todo el dominio del espacio.

3. **Búsqueda telescópica:** No sólo podemos reducir el área de búsqueda, reduciendo la resolución de cada imagen, sino siendo conscientes de las características inherentes que todo vídeo tiene (probabilísticamente hablando). Esto es, que los campos de movimiento de dos imágenes consecutivas probablemente sean muy similares. Y aprovechando esta similitud, el campo de movimiento del segundo frame se pueda calcular mas rápidamente si partimos del campo de la imagen anterior, para posteriormente ir especificando con mas exactitud el vector de movimiento concreto para dicho frame en búsquedas muy localizadas. Así las técnicas vistas anteriormente pueden ser mas rápidas si la búsqueda se realiza en un área reducida y suponiendo que los vectores de movimiento no van a cambiar bruscamente de ubicación entre frames.

III-C3. El predictor ($\mathcal{P}$): El predictor usado en MPEG-1 está basado en la copia (P) o promediado (B) de macro-bloques extraidos de la(s) imágen(es) de referencia a partir de los vectores de movimiento determinados en la fase ME (véase la Sección III-C2). En la Figura 9 se muestra un ejemplo de este proceso.

III-C4. La transformada discreta del coseno ($\mathcal{T}$): La transformación es idéntica a la usada en JPEG (véase la Sección III-A4).

III-C5. El cuantificador ($\mathcal{Q}^q$): Se trata de un proceso de cuantificación escalar idéntico al usado en JPEG (véase la Sección III-A5).

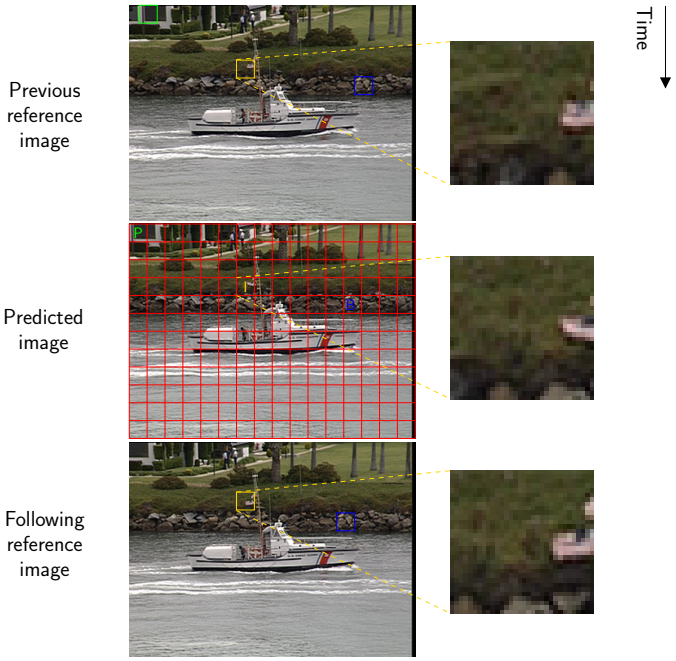

Figura 9. El proceso de predicción en MPEG-1.

Motion VLC code	offset		
0000 0011 001	−16	1	0
0000 0011 011	−15	010	1
0000 0011 101	−14	0010	2
0000 0011 111	−13	0001 0	3
0000 0100 001	−12	0000 110	4
0000 0100 011	−11	0000 1010	5
0000 0100 11	−10	0000 1000	6
0000 0101 01	−9	0000 0110	7
0000 0101 11	−8	0000 0101 10	8
0000 0111	−7	0000 0101 00	9
0000 1001	−6	0000 0100 10	10
0000 1011	−5	0000 0100 010	11
0000 111	−4	0000 0100 000	12
0001 1	−3	0000 0011 110	13
0011	−2	0000 0011 100	14
011	−1	0000 0011 010	15
		0000 0011 000	16

Figura 10. Códigos de longitud variable usados para codificar los vectores de movimiento en MPEG-1.

III-C6. Codificación de las texturas residuo (VLC): De nuevo, el MPEG-1 hereda esta funcionalidad del JPEG (véase la Sección III-A6).

III-C7. Codificación de los vectores de movimiento (VLC): En MPEG-1, los vectores de movimiento se codifican usando el código de Huffman mostrado en la Figura 10. Dichos vectores son representados como una tupla *(x,y)*, donde *x* e *y* siguen una distribución de probabilidad de Laplace con un rango de $0 \leq x, y < 16$. Se trata de un código de longitud variable para el desplazamiento del movimiento en otras palabras, el código de compresión para un determinado vector de movimiento.

III-C8. En controlador del bit-rate (BC): MPEG-1 posee básicamente dos modos de funcionamiento:

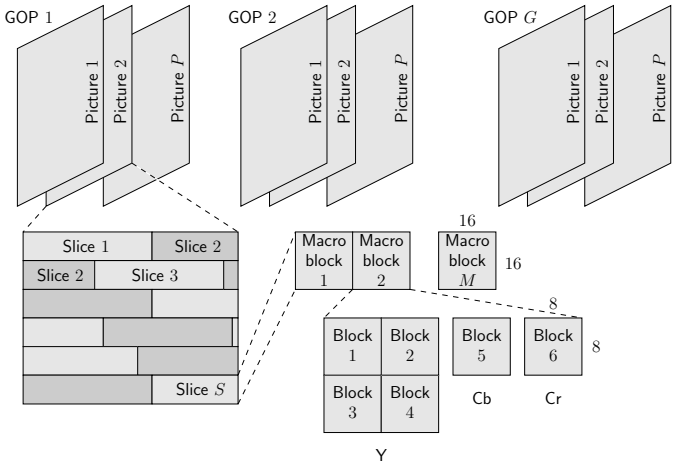

Figura 11. Distribución de los datos en MPEG1.

1. **Modo CBR (Constant Bit-Rate):** En este modo, el módulo BC decide qué niveles de cuantificación deben ser utilizados para que la tasa de bits de salida del compresor permanezca constante a lo largo del tiempo.
2. **Modo VBR (Variable Bit-Rate):** Por contrapartida, en este modo el sistema BC no funciona. El nivel de cuantificación se mantiene constante y por tanto, la cantidad de bits a la salida es proporcional al contenido visual de la secuencia de vídeo.

III-C9. El multiplexor de texturas/movimiento (Mux): Una vez que los code-streams de las texturas y el movimiento han sido generados, llega el momento se entrelazarlos para que durante la reproducción, tras un cierto retardo generalmente inapreciable, ambos streams puedan ser descomprimidos simultáneamente. Esta tarea de entrelazamiento la realiza el multiplexor.

III-C10. Distribución de los datos en MPEG-1: Cada una de las capas (slice en la imagen) es codificada independientemente quedando comprimida y es etiquetada con un código especial en cadena. Esto aumenta la capacidad de recuperación de errores del código. El tamaño mínimo de una capa es un macro-bloque y el máximo es un frame completo.

Como se observa en la Figura 11 cada macro-bloque contiene bloques de datos de luminancia y crominancia (cuatro bloques de luminancia, Y1, Y2, Y3, Y4, y dos de crominancia, U y V).

III-D. MPEG-2

El nombre de estándar es "Generic Coding of Moving Pictures and Associated Audio" o *ISO/IEC 13818-1*. Creado en 1994, se masificó su uso en DVDs (Digital Versatile Disc), televisión digital y DVB-T (Digital Video Broadcasting Terrestrial). Constituye una evolución de MPEG-1, el cual, se puede considerar como un subconjunto del MPEG-2 siendo compatible con éste. Sus ventajas son: el soporte para audio multicanal, vídeo entrelazado (el formato usado por las televisiones) y progresivo (el usado en los motitores de ordenador, cámaras fotográficas y de vídeo, etc.).

La sintaxis del MPEG-2 tiene dos categorías:

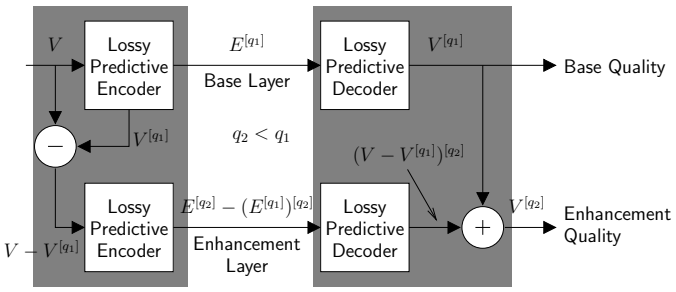

Figura 12. Escalabilidad en calidad MPEG-2.

1. Una sintaxis *no escalable*, la cual incluye a la sintaxis del MPEG-1, con extensiones adicionales para soportar vídeo entrelazado.
2. Una sintaxis *escalable*, que permite una codificación por capas de la señal de vídeo, mediante la cual, el descodificador puede descodificar:
   - Sólo la capa básica para obtener una señal con calidad mínima, o
   - utilizar capas adicionales para incrementar la calidad de la señal. Lo que lo hace ideal para entornos de visualización remota, ya que se puede adaptar el peso de la señal enviada según este parámetro.

En comparación MPEG-2 es mejor que su antecesor en que:

1. **Posee un mayor rango de bit-rates.** Concretamente entre 4 Mbps y 30 Mbps. MPEG-2 no esta optimizado para bajas tasas de bits (menores que 1 Mbit/s), pero supera en desempeño a MPEG-1 a 3 Mbit/s y superiores.
2. **Soporta mayor resolución.** Es el estándar de televisión en alta definición (HDTV).
3. **Posee una resistencia mejorada a los errores.** MPEG-2 introduce y define *Flujos de Transporte*, los cuales son diseñados para transportar vídeo y audio digital a través de medios impredecibles e inestables (como es la TDT o la TV por satélite). Además como se expondrá en el siguiente punto la escalabilidad SNR también contribuye a esta mejora.[2]
4. **Es escalable a nivel SNR y espacial.**
   a) En la Figura 12 se ilustra el mecanismo que utiliza MPEG-2 para llevar a cabo la **escalabilidad SNR**, donde se dibuja a grandes rasgos el proceso de codificación y decodificación con pérdida, propio de MPEG-2. La notación usada es la misma que en los anteriores gráficos. La "cantidad de calidad" de la imagen viene determinada por el bit-rate que se le asigna y por el número de capas que la compongan. Se representa por un superíndice en

[2]El hecho de que el esquema de codificación de vídeo MPEG-2 utilice técnicas de eliminación de la redundancia o compresión hace que cualquier aplicación de vídeo MPEG-2 sea muy vulnerable ante la pérdida de información y errores en la decodificación o en la transmisión. Su impacto negativo sobre la calidad de vídeo se ve amplificado debido a los fenómenos de propagación espacial y temporal de errores presentes en el estándar MPEG-2. Puesto que no hay un esquema adicional que controle la propagación de las pérdidas o los errores. Por tanto, la escalabilidad permite introducir códigos de detección y corrección de errors de forma desigual, haciendo mucho más resistente la capa básica que la de mejora y de esta forma, el bit-rate efectivo no se ve incrementado de forma significativa.

la Figura 12, en el que a mayor índice, menor bit-rate, de tal forma que $V_i^{[0]} = V_i$ (siendo V_i la i-ésima imagen de la secuencia de vídeo).
Continuando con la Figura 12, se observa que el flujo de bits es dividido en dos capas. La idea esta basada en el hecho de que el sistema visual humano es más insensible a las componentes de alta frecuencia de las señales de vídeo. Las semejanzas entre capas consisten en que tienen la misma resolución espacial para diferentes calidades de vídeo. Así como que cada capa contiene todos los coeficientes DCT, pero con la particularidad de que cada capa puede tener un valor diferente del factor de escala. A continuación se describen los tipos de capas: La llamada "capa base" o **base layer** transmite con un nivel de prioridad alta: los coeficientes de baja frecuencia, los modos de codificación y los vectores de movimiento. La "capa de mejora" o **enhancement layer**, transmite en un nivel de prioridad mas bajo donde la probabilidad de pérdidas es mayor: Los coeficientes de alta frecuencia, entre otra información menos importante, proporcionando un mayor refinamiento de los coeficientes de la DCT de la capa básica, mejorando la calidad proporcionada por ésta. Así la escalabilidad SNR proporciona también una mejor resistencia a errores de transmisión. Por ejemplo, las capas de mejora pueden transmitirse sobre canales con peores prestaciones, mientras que la capa básica (sobre mejores medios) provee siempre de un mínimo de la señal.
Este modo de escalabilidad es idóneo en aplicaciones de transmisión de vídeo, cuando se tienen redes de transmisión que soportan dos niveles de prioridad como las redes con tecnología ATM [9].

b) La **escalabilidad espacial** fue diseñada para ser utilizada como herramienta que permitiese la interoperatividad entre diferentes estándares y entre diferentes sistemas con diferentes resoluciones como dispositivos que trabajan a bajas resoluciones (un móvil, por ejemplo), resoluciones medias (una TV, por ejemplo), y altas resoluciones (la HDTV, por ejemplo), con un mismo flujo de bits (pudiendo tener diferentes: tamaños de cuadro, caudal de cuadros y formatos de muestro).
El *flujo de bits* se divide en capas de diferente resolución espacial. La capa básica es codificada por si misma y proporciona la resolución espacial básica, mientras que la capa de mejora utiliza la capa básica, para proporcionar la resolución espacial completa de la señal de vídeo.

III-E. MPEG-4

La cuarta y última versión de MPEG esta disponible desde 1999 y es muy usado en contenido DivX. Su nombre técnico es *Coding of audio-visual objetcs* [10]. Como su anterior versión también esta basado en una codificación híbrida, pero ha sido optimizado para trabajar con niveles de bit-rate muy bajos [11], hasta 5 Kbps.

Los medios audio-visuales son descompuestos en objetos (objetos visuales y de audio), donde cada objeto puede ser sintético o natural, y puede ser codificado usando diferentes técnicas como sprites, codificación híbrida, mallas 2D ó 3D con texturas, etc.

Las ventajas de MPEG-4 frente a la versión anterior (MPEG-2) se resumen básicamente en refinar la precisión a 1/4 de sub-pixel. Aumentar el número de vetores de movimiento o macro-bloques así como de referencias a imágenes hasta 4. Y mejorar la corrección de errores en el código.

III-E1. Niveles y perfiles: La mayoría de las características que conforman el estándar MPEG-4 no tienen que estar disponibles en todas las implementaciones, al punto que no existen actualmente implementaciones completas del estándar MPEG-4 [12]. Para manejar esta variedad, el estándar incluye el concepto de *perfil* (profile) y *nivel*, lo que permite definir conjuntos específicos de capacidades que pueden ser implementados para cumplir con objetivos particulares. Los principales parámetros de cada perfil son:

1. Tamaño típico de visionado.
2. Máximo número de objetos.
3. Máximo bit-rate (Kbps).
4. Máximo número de capas de mejora.

III-E2. Mayor precisión en la escalabilidad de la granularidad: Debido a la amplia variación de ancho de banda disponible en intervalos cortos para sesiones inalámbricas, hay una necesidad de métodos de codificación de vídeo escalable que permitan al flujo de datos adaptarse flexiblemente a las condiciones de red cambiante en tiempo real. Una técnica es la *Escalabilidad Granular Fina (FGS)* de MPEG-4, la cual puede adaptarse en tiempo real a variaciones de ancho de banda mientras usa el mismo flujo de datos pre-codificado. Por tanto, las ventajas clave del entramado de MPEG-4 FGS (elasticidad y flexibilidad) vienen a expensas de una calidad de vídeo sensiblemente inferior.

III-F. H.264/AVC

El estándard H.264/AVC [13], es también llamado *Estandar ISO/IEC 14496-10*. Constituye una evolución más de MPEG-4, siendo de hecho su parte número 10. Es una norma que define un códec de vídeo de alta compresión, desarrollada conjuntamente por el ITU-T Video Coding Experts Group (VCEG) y el ISO/IEC Moving Picture Experts Group (MPEG). La intención del proyecto H.264/AVC fue la de crear un estándar capaz de proporcionar una buena calidad de imagen con tasas binarias notablemente inferiores a los estándares previos (MPEG-2, H.263 o MPEG-4 parte 2), además de no incrementar la complejidad de su diseño.

III-F1. Diferencias con los codecs anteriores: Para empezar a programar el código del nuevo estándar se adoptaron las siguientes premisas:

1. La estructura DCT + Compensación de Movimiento de las versiones anteriores era superior a otros estándares y por esto no había ninguna necesidad de hacer cambios fundamentales en la estructura.

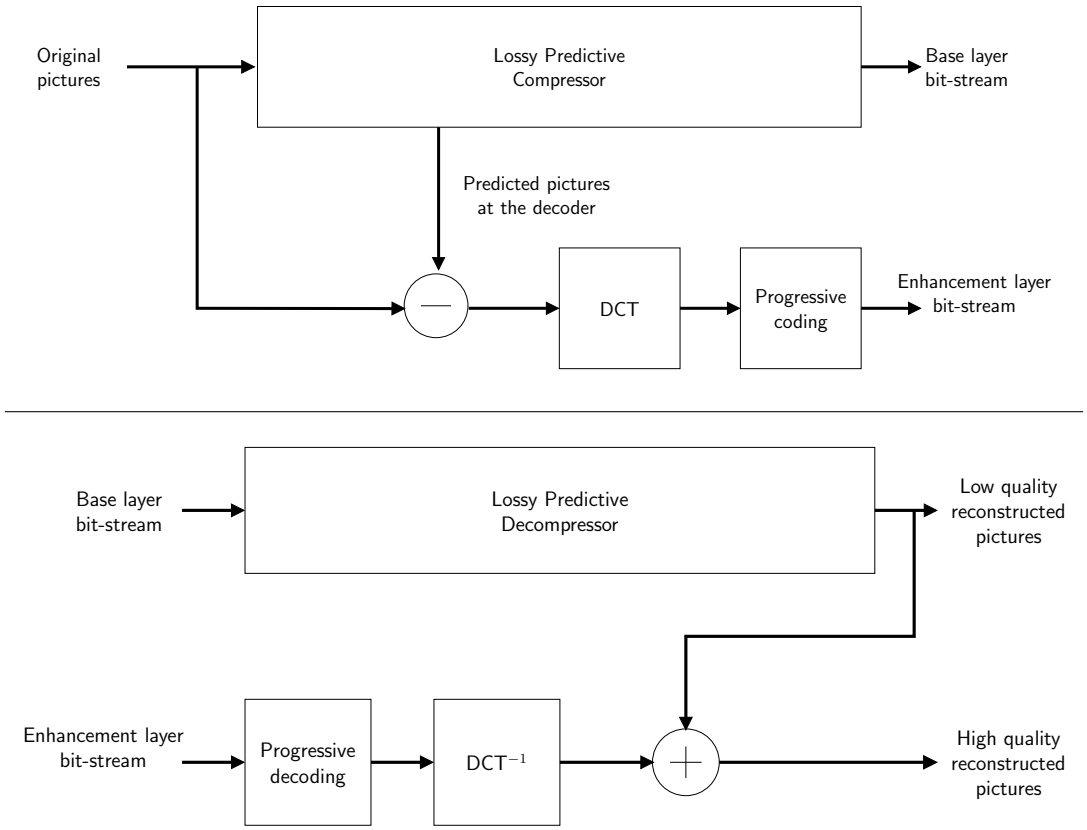

Figura 13. El codec MPEG-4 escalable en calidad. Arriba el compresor y debajo el descompresor.

2. Algunas formas de codificación de vídeo que habían sido excluidas en el pasado debido a su complejidad y su alto coste de implementación se volvieron a examinar para su inclusión puesto que la tecnología VLSI *Very Large Scale Integration* había sufrido un adelanto considerable y una bajada de costes de implementación.
3. Para permitir una libertad máxima en la codificación y evitar restricciones que comprometan la eficiencia, no se contempla mantener la compatibilidad con normas anteriores.
4. El tamaño de los macro-bloques puede ser de 16, 8 y 4, en cualquier combinación.
5. Codificación mejorada de la entropía:

   a) CAVLC (Coded according to the context adaptive variable length) [14]. El objetivo de esta codificación es procesar la información que se quiere transmitir o almacenar en un dispositivo de forma que ocupe el mínimo espacio posible. De esta manera, con el uso de la CAVLC sera posible transmitir una imagen en menos tiempo o hacer que ocupe menos espacio en el dispositivo de almacenamiento. Una característica importante de esta codificación es que no tiene pérdidas y que por lo tanto se podrá recuperar la información original al aplicar el proceso inverso. Se emplea para codificar y comprimir la información que resulta de la aplicación de la transformación y cuantificación de un bloque de luminancia de tamaño 4x4 píxeles.

   b) CABAC, en inglés Context Adaptative Binary Arithmetic Coder. Presenta una mejora en la dimensión del bitstream respecto a CAVLC, dado que consigue bitstreams un 10 por ciento mas pequeños. La arquitectura de CABAC demuestra que es bastante buena ya que puede comprimir eficientemente la señal original. Sin embargo, el rendimiento se podría mejorar considerando que cada coeficiente es estadísticamente dependiente de sus vecinos.
6. Soporte de hasta 16 referencias a imágenes en muestras de macro-bloques tipo B.
7. Post-compresión que mejora los ratios conseguidos inicialmente.
8. Compresión sin pérdida, si no se lleva a cabo la cuantificación.

III-F2. Perfiles: El uso inicial del MPEG-4/AVC estuvo enfocado hacia el vídeo de baja calidad (ej. videoconferencia) basado en 8 bits/muestra y con un muestreo ortogonal de 4:2:0. Esto no daba salida al uso de este códec en ambientes profesionales que exigen resoluciones más elevadas, necesitan más de 8 bits/muestra y un muestreo de 4:4:4 ó 4:2:2, funciones para la mezcla de escenas, tasas binarias mas elevadas, poder representar algunas partes de vídeo sin pérdidas y utilizar el sistema de color por componentes RGB. Por este motivo surgió la necesidad de una extensiones, llamadas ”extensiones de gama de fidelidad” *(FRExt)* que incluían soporte para:

1. Un tamaño de transformada adaptativo,

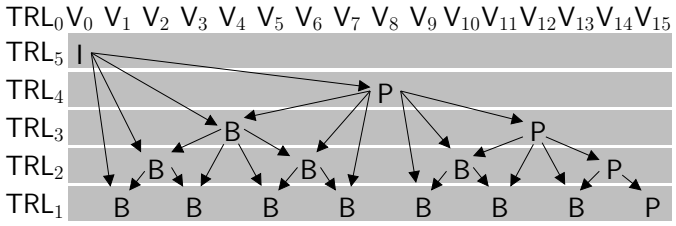

Figura 14. Escalabilidad temporal usando MCTF en H.264/SVC.

2. una cuantificación con matrices escaladas y
3. una representación eficiente sin pérdidas de regiones específicas. El conjunto de extensiones denominadas de "perfil alto" soportan 4:2:0 con hasta 8 bits/muestra, 4:2:0 ó 4:2:2 con hasta 10 bits/muestra y, por último, 4:4:4 con 12 bits/muestra y la codificación de regiones sin pérdidas.

III-G. H.264/SVC

También referenciado como *Estandar ISO/IEC 14496-10*, continua en desarrollo. Consiste en la versión escalable de H.264/AVC, siendo compatible con éste. En otras palabras, la capa base dada por el codec H.264/SVC puede se descomprimida por H.264/AVC.

III-G1. Diferencias con los codecs anteriores: Aunque H.264/SVC es compatible hacia atrás con H.264/AVC, define un nuevo esquema de descorrelación temporal llamado MCTF [15], en inglés *Motion Compensated Temporal Filtering*. Es más eficiente que su homólogo anterior (recordemos el esquema IPPP...), además de permitir la escalabilidad en el tiempo.

H.264/SVC no sólo aporta esta nueva forma de escalabilidad, sino que también las otras dos, la espacial y en calidad. Así el diseño del SVC apoya la creación de flujos de bits de calidad *escalable* que se pueden convertir en flujos de bits que se ajustan a uno de los perfiles H.264/AVC *no escalable* mediante una reescritura del proceso a baja complejidad.

III-G1a. Escalabilidad temporal: La escalabilidad temporal provee de un acceso rápido a cualquier frame del vídeo. Concretamente en H.264/SVC se implementa la escalabilidad temporal por medio de MCTF. En la Figura 14 se representan las dependencias entre distintos tipos de imágenes que se crean para permitir la escalabilidad temporal en este codec.

En el *eje X* figuran los tipos de imágenes (I, P y B). Recordemos que V_0 se corresponde con una imagen *intra (I)* la cual tiene la misma codificación que la imagen original sin comprimir y sin vectores de movimiento. En el *eje Y*, TRL, representa los *niveles de resolución temporales*, o en otras palabras, los frames por segundo. Es interesante observar que, según la capacidad del medio o por petición expresa, el codec es capaz de mostrar el número de imágenes por segundo adecuado, de tal forma que, si se desea sólo una imagen, se muestra la intra, si se desean 2, se muestra la I y la P (la imagen P depende de la I), si se desean 3, se muestran la I, dos P y una B (con sus respectivas dependencias). Según el esquema del ejemplo presentado, la escalabilidad temporal podría proveer desde 1 hasta 8 imágenes por segundo.

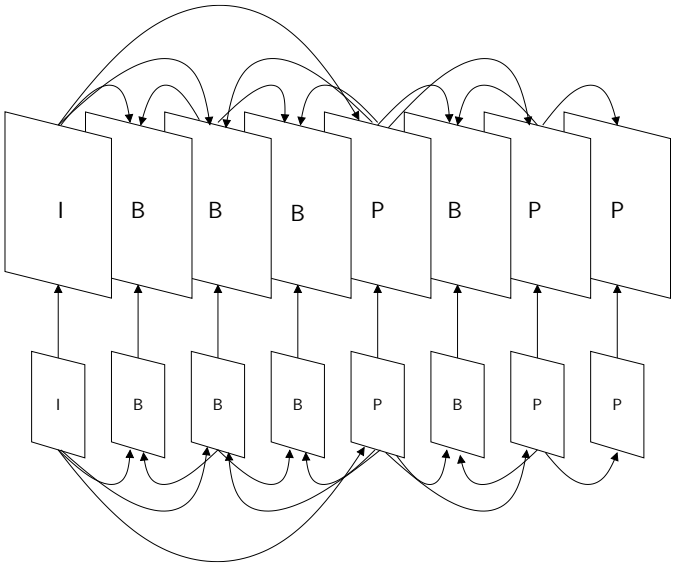

Figura 15. Escalabilidad espacial en H.264/SVC.

Capa (D)	Resolución	Calidad	Ratio de la imagen	(S,T,Q)
0	176x144	0	3.75	(0,0,0)
1	176x144	0	7.5	(0,1,0)
2	176x144	0	15	(0,2,0)
3	176x144	0	30	(0,3,0)
4	176x144	1	3.75	(0,0,1)
5	176x144	1	7.5	(0,1,1)
6	176x144	1	15	(0,2,1)
7	176x144	1	30	(0,3,1)
8	352x288	0	3.75	(1,0,0)
9	352x288	0	7.5	(1,1,0)
10	352x288	0	15	(1,2,0)
11	352x288	0	30	(1,3,0)
12	352x288	1	3.75	(1,0,1)
13	352x288	1	7.5	(1,1,1)
14	352x288	1	15	(1,2,1)
15	352x288	1	30	(1,3,1)

Figura 16. Ejemplo de generación de capas en H.264/SVC.

III-G2. Escalabilidad espacial: Como se observa en la Figura 23 en comparación con la anterior (Figura 14). Las dependencias entre imágenes para las escalabilidades temporal y espacial, son evidentemente distintas. Por poner un ejemplo: la imagen intra (I) sirve de base para dos B y una P. Si bien se exponen las dependencias entre imágenes en un mismo rango en el dominio del espacio menor (abajo), también vemos que cada imagen se corresponde de forma directa con su versión en un rango del espacio mayor (arriba).

III-G3. Escalabilidad en calidad: Las capas son identificadas por su dependencia con el identificador D (véase le Figura 16). Cada capa contiene una o mas capas de calidad, que representan la fuente de vídeo para un instante de tiempo con una resolución espacial y una fidelidad (calidad) específica. Éstas se representan en la última columna de la tabla con las siglas *T*, *S* y *Q* respectivamente.

El conjunto de capas de calidad dependientes entre sí corresponden a la misma resolución espacial y se identifican por un identificador de calidad Q. Las capas de calidad son creadas progresivamente [16] según un nivel de cuantificación decreciente. De este modo, para las capas de refinamiento de la calidad con valor de $Q > 0$ indica que siempre la capa

anterior de calidad, con un identificador de calidad Q - 1, se emplea para la capa intra de predicción. Y para el valor de Q = 0, se puede seleccionar como capa de referencia para la predicción cualquier capa anterior. De ella se extraen además las siguientes afirmaciones:

1. Si la capa de mayor calidad, la número 15, no puede ser descomprimida por cualquier motivo, por ejemplo, falta de ancho de banda del medio, entonces las imágenes impares deberán ser descomprimidas con la mínima calidad.
2. Si la capa número 14 se pierde, entonces sólo las imágenes cuyo índice sea divisible entre 4 se obtendrán con la máxima calidad.
3. En caso de que las capas 12 a 15 se pierdan, entonces todas las imágenes se obtendrán con calidad mínima.
4. Si la capa 11 se pierde, entonces las imágenes impares se obtendrán con calidad 1 pero con una resolución de 176x144.
5. Si sólo se descomprime la capa base, la 0, entonces sólo una imagen de cada 8 se visualizará, con calidad 0 y un tamaño de 176x144.

III-G4. Número de capas: Bajo un escenario real de transmisión, como puede ser Internet, la cantidad de datos que se pueden transmitir dependen de múltiples factores impredecibles y esto debe prevenirse en la compresión del vídeo. Para maneja esta situación, el número de capas de calidad debe ser lo mas alto posible para así poder adaptarse a la red y aprovechar al máximo el ancho de banda. Ya que no se puede transmitir media capa, en cuantas mas capas se divida la calidad, con mayor precisión se ajustara a los requisitos de la red. Sin embargo, se debe llegar a un compromiso para dicho número, puesto que a mayor nº de capas menor ratio de compresión.

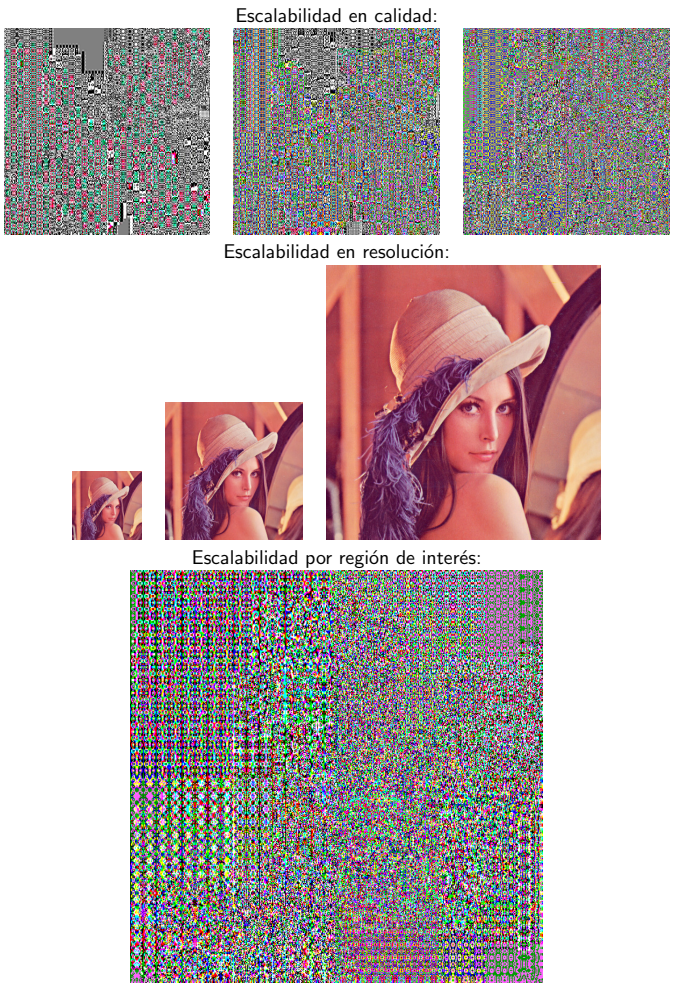

Figura 17. Las tres formas de escalabilidad en JPEG 2000.

III-H. JPEG 2000 (J2K)

JPEG 2000 es el último estándar de compresión de imágenes desarrollado por la ISO [17]. Fue creado con la idea de reemplazar al omnipresente estándar JPEG. Como puede apreciarse en la Figura 18, el compresor JPEG 2000 (sólo J2K en adelante) está formado por una serie de etapas secuenciales que son:

1. La transformada de color, que eliminar la correlación ínter-componente.
2. La transformada espacial (una trasformada wavelet), que elimina la correlación intra-componente.
3. La fase de cuantificación que elimina la información visualmente menos relevante.
4. La etapa de definición de la(s) ROI(s) (Regions Of Interest).
5. La codificación de longitud variable o VLC (Variable Length Coding) que hace efectiva la compresión de los datos.
6. La etapa PCRD-opt [18] (Post Compression Rate-Distortion Optimization), que se utiliza para organizar los datos en el code-stream con el objeto de conseguir las diferentes formas de escalabilidad.

J2K presenta sobre JPEG las siguientes ventajas:

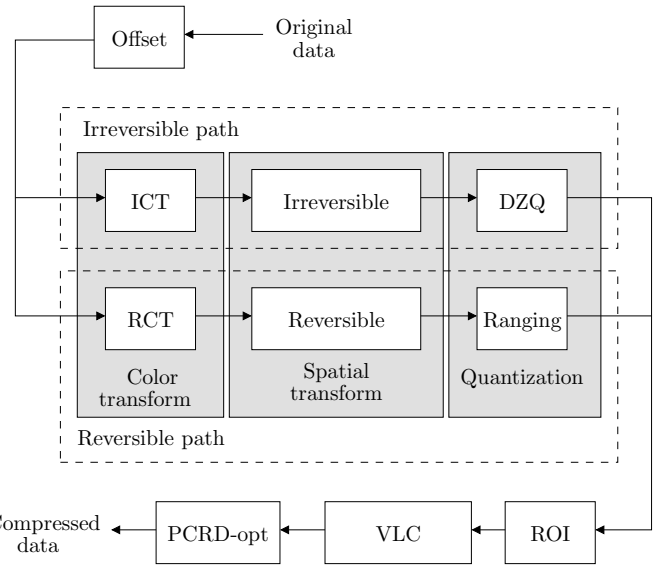

Figura 18. Arquitectura del compresor JPEG 2000.

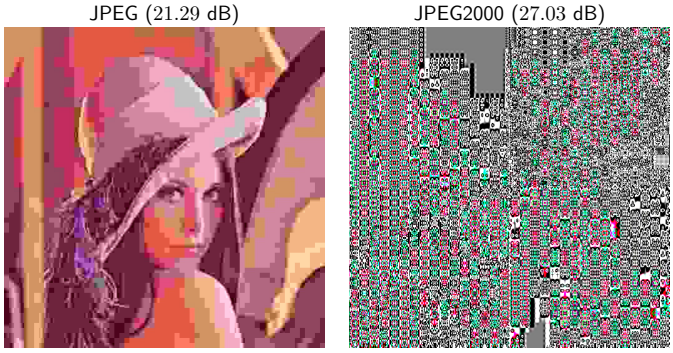

Figura 19. La imagen Lena comprimida a 0.1 bits/pixel usando JPEG (izquierda) y JPEG 2000 (derecha). La calidad visual se ha expresado además usando la métrica PSNR[dB].

1. Es más eficiente en términos R/D (véase la Figura 19).
2. Pueden realizarse tanto compresiones *lossy* (irreversibles) como *lossless* (totalmente reversibles), dependiendo de los requerimientos del usuario. La versión lossy arroja tasas de compresión ligeramente superiores a las de la versión lossless.
3. Es escalable en calidad, en resolución y en región y ventana[3] de interés. En la Figura 17 se muestra un ejemplo de cada tipo de escalabilidad. Para conseguir estas tres formas de escalabilidad a partir de un único code-stream, el estándar define (entre otras) tres ordenaciones diferentes de los paquetes de datos JPEG 2000 [19]. La progresión LRCP (Layer-Resolution-Component-Precinct) permite una descodificación progresiva en calidad al ordenar los datos referentes a los coeficientes wavelet por planos de bits. La progresión RLCP produce una descodificación progresiva por niveles de resolución al ordenar los datos referentes a los coeficientes wavelet por subbandas de frecuencia espacial incremental y finalmente, la progresión por WOI (Window Of Interest) se genera al poder seleccionar el subconjunto correspondiente de paquetes que hacen referencia a dicha ventana.

III-I. Motion JPEG 2000 (MJ2K)

Motion JPEG 2000 [20] se define en la norma *ISO / IEC 15444-3* y en la *UIT-T T.802*. En ella se especifica el uso del codec JPEG 2000 para las secuencias de imágenes programadas (secuencias de movimiento), combinado con audio, y compuesto en una presentación global. Las extensiones de archivos para archivos de vídeo Motion JPEG 2000 son `.mj2` y `.mjp2` de acuerdo con la norma *RFC 3745*. Es el principal soporte de cine digital estándar en la actualidad con el apoyo de las *Digital Cinema Initiatives*, un consorcio de la mayoría de los principales estudios y proveedores, para el almacenamiento, distribución y exhibición de películas cinematográficas. También se está considerando como un formato de archivo digital por la Biblioteca del Congreso.

[3]Cuando el área de la imagen a descomprimir con mayor calidad se especifica durante la compresión se habla de región de interés o ROI (Region Of Interest). Una ROI puede tener cualquier forma posible. Sin embargo, cuando el área se define durante la descompresión, nos referimos a una ventana o WOI (Window Of Interest), que sólo puede tener forma rectangular.

A diferencia de los codecs de vídeo comunes, como MPEG-4, MJ2K no emplea la compresión de estructura temporal. En su lugar, cada fotograma es una entidad independiente codificada por una variante con o sin pérdidas de JPEG 2000.

III-J. Motion Compensated Temporal Filtering (MCTF)

El estudio de nuevas formas de mejorar la codificación de vídeo y desarrollo de las ya existentes es muy prolífico. Cada uno de esos trabajos consiguen buenos resultados para un subconjunto de características que debería tener la codificación ideal para un vídeo. Y aunque en algunos trabajos se encuentren formas de mejorar alguna característica concreta, como por ejemplo, una rápida accesibilidad aleatoria al vídeo [21], [22], o simplemente hacer el vídeo mas portable en una transmisión por un canal estrecho [23], [24]. Sin embargo, es evidente que la compensación temporal de las imágenes de un vídeo reúne el mayor número de ventajas, entre ellas, reduce notablemente los Kbps necesarios para codificarlo. Por ello es un planteamiento muy usado. Sin embargo, aun no existe una metodología que estandarice todo el proceso. De este modo existen numerosos estudios que aplican algún tipo de predicción del movimiento, con técnicas del tipo MCTF, como LIMAT, en inglés *Lifting-based Invertible Motion Adaptive Transform* y que presentan resultados interesantes.

En investigaciones sobre LIMAT [25] se ha trabajado con mallas deformables que pueden mejorar la compensación de movimiento debido a su adaptabilidad a las condiciones de su contexto (particularidades de la secuencia de imágenes), gracias a las expansiones y contracciones en el marco de seguimiento de los movimientos en la imagen, en comparación con mantener un campo de movimiento continuo. Y se concluye que el entorno de trabajo de LIMAT, con el uso de la DWT tiene un rendimiento superior, al menos de 5/3 sobre sólo 1/3 utilizando la transformada de Haar. En esta misma línea de estructuras más flexibles, se estudió la aplicación de DWT para la escalabilidad espacial [7], [26], [27] y como aprovecharlo con el fin de minimizar cálculos para la predicción. Así como también, llevar a cabo un aumento progresivo de la precisión de las referencias para la predicción del movimiento [8].

En el Instituto de Heinrich Hertz (Alemania) se desarrolló la norma H.264, ya comentada, y con ella, el sistema de estimación del movimiento de MCTF. Existen implementaciones de MCTF [28] que usan múltiples frames para el seguimiento del movimiento. De modo que el margen en que se pueden detectar macro-bloques similares se mayor y, por tanto, la predicción es más amplia.

Pero este planteamiento no carece de por menores y algunos trabajos se enfocan sobre los problemas que conlleva aplicar la correlación temporal de imágenes, buscando un compromiso entre eficiencia y calidad de visualización. Uno de los problemas más llamativos es una distorsión visualmente parecida al llamado *efecto fantasma* (ver Figura 20), producido precisamente por el simple hecho de relacionar dos o más imágenes entre sí y dar como resultado fotogramas en los que se muestra información de varios instantes de tiempo a la vez. Esta línea se toma en trabajos donde se consigue reducir este ruido mediante técnicas aplicadas al paso baja de sub-

Figura 20. Datos cruzados de varias imágenes.

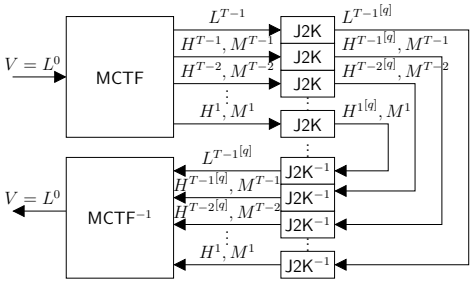

Figura 21. Estructura básica de MCJ2K.

bandas temporales [29], mejorando además las escalabilidades temporal y espacial.

Otro planteamiento consiste en el concepto de imágenes clave [16] para controlar eficazmente la deriva de la calidad en la codificación basada en paquetes escalables con las estructuras jerárquicas de predicción. O incluso ampliando el procesado a las partes de la imagen que no se encuentren en movimiento, usando un método de adaptación de contenido para compensación de movimiento en tres dimensiones [30].

Como este tipo de trabajos, muchos otros consisten en aumentar la calidad de imagen mediante procesos adicionales al compensado temporal. Desde otro punto de vista, existen los trabajos que mediante una *Highly scalable video compression* [31] consiguen optimizar cuantitativamente el número de cálculos de los procesos basados en la Transformada Wavelet. Aumentando significativamente el rendimiento con un detrimento relativamente pequeño en calidad.

III-K. Motion Compensated JPEG 2000 (MCJ2K)

III-K1. El papel de MCTF en MCJ2K: Como se ha visto MCTF elimina eficientemente la redundancia temporal de una secuencia de imágenes [32] y provee de una escalabilidad temporal y controles para minimizar el impacto de errores y pérdidas en los datos.

De igual modo, como todos sabemos, J2K es un buen compresor de imágenes y puede crear codificaciones escalables en espacio y calidad. Es posible que ambos codecs pueden utilizarse juntos para diseñar un compresor escalable en espacio, tiempo y calidad. El esquema presentado en la Figura 21 es simple y constituye el esqueleto de MCJ2K.

III-K2. MCTF en MCJ2K: En MCJ2K los GOPs son siempre abiertos y simétricos en cada resolución temporal. Como puede verse en la Figura 22 se presentan 2 GOPs, el primero esta compuesto únicamente por V_0 y el segundo lo constituyen de V_1 a V_{16}.

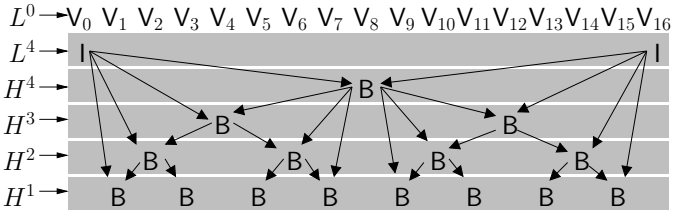

Figura 22. MCTF con suficiente correlación temporal.

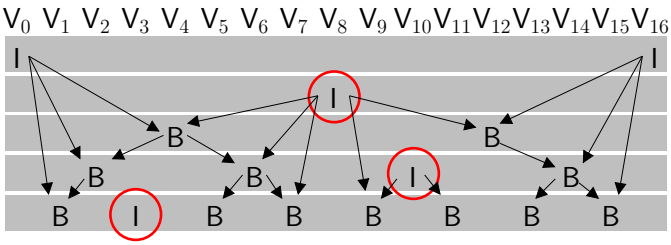

Figura 23. MCTF sin suficiente correlación temporal.

En el caso en que no se encuentre suficiente correlación temporal, las imágenes B son reemplazadas por imágenes I (y las referencias de los vectores de movimiento se eliminan). Tal y como se observa en la Figura 23. Dicho con otras palabras, una imagen B puede finalmente sustituirse por una INTRA si la entropía de B es demasiado grande (siendo igual o mayor). Es decir, la imagen que se desea predecir mediante vectores de movimiento, es tan distinta a la anterior, que es menos costoso codificar la imagen directamente como INTRA, en lugar de enumerar sus diferencias.

La implementación de MCTF debe sufrir ciertas modificaciones para adaptarse al esquema inicial en la Figura 21. Aquí MCTF sigue una típica implementación recursiva ”filter bank”, con transformaciones tanto hacia adelante como hacia atrás, representada en la Figura 24.

Podemos ver en detalle en la Figura 25 uno de los compenentes de transformación de nivel temporal directo e inverso, propio de las cajas de la Figura 24 desde un punto de vista a más alto nivel.

En la Figura 25, el superíndice *t* representa el nivel de transformación Wavelet, y el subíndice *i* el número de fotograma al que pertenece la muestra. En esta línea observamos que se da

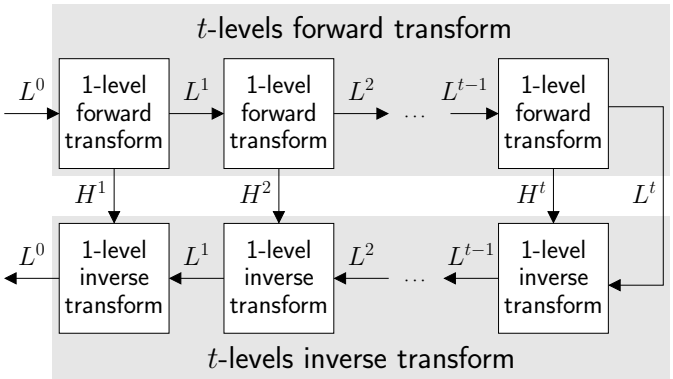

Figura 24. Transformación en el nivel temporal.

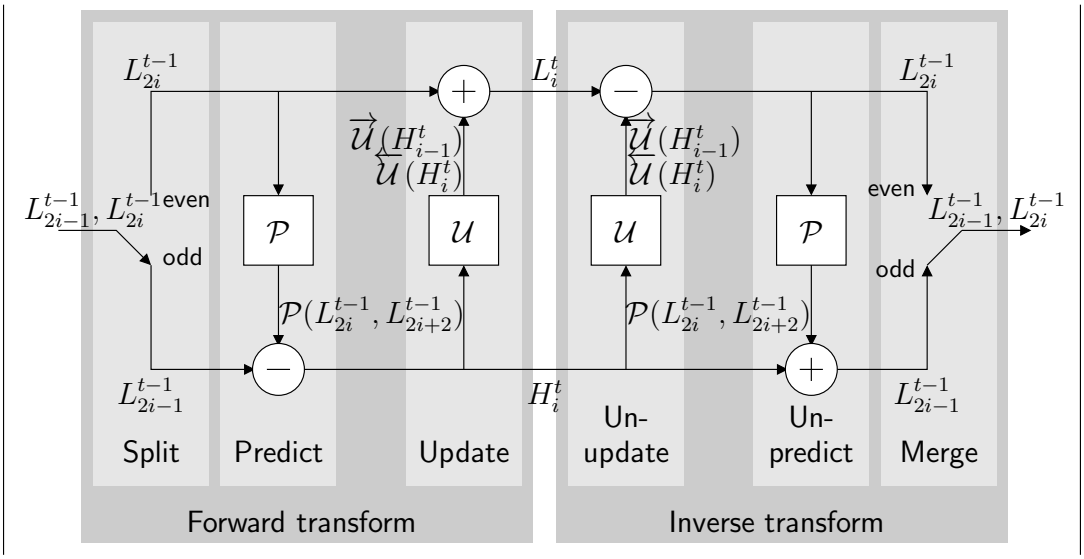

Figura 25. Nivel de transformación directo (izquierda) e inverso (derecha).

un trato distinto a los fotogramas pares e impares, donde en los primeros se lleva a cabo la predicción y en los segundos se compara la diferencia entre lo predicho y lo que realmente ocurre.

El mecanismo de una transformada Wavelet consiste en dividir recursivamente los promedios y las diferencias, llevar a cabo una predicción del movimiento (mediante el establecimiento de referencias o vectores de movimiento) y finalmente una actualización o refinamiento de dichas referencias. La transformada inversa consiste en llevar a cabo los pasos en orden inverso, para finalmente unir las medias y las diferencias. Por ejemplo, sean dos señales A y B, promedio S y diferencia D. Se puede recuperar A y B a partir de S y D, como ecuación. También se puede invirtiendo una matriz 2x2, de ahí la llamada transformada Wavelet de Haar [33]. A mayor correlación entre las dos señales mas pequeña sera D (la resta entre ambas, su diferencia) y se podra representar con menos bits.

III-K3. Escalabilidad temporal, espacial y en calidad: Lo que hace posible estas tres escalabilidades es la división de los datos en capas y la organización de éstas. Existen 5 tipos de ordenaciones (dos muy utilizados) para almacenar los paquetes que componen un fotograma o imagen de un vídeo. Dado que los paquetes en sí no modifican, su ordenación se puede cambiar de un tipo a otro. Claro esta, siempre y cuando dispongamos de dichos paquetes.

La **escalabilidad temporal** representa el número de imágenes por segundo que se transmiten, es decir, la cantidad de paquetes enviados para la resolución y calidad. Para conseguir que funcione correctamente las sub-bandas temporales deben ser decodificadas en orden, para visualizar la secuencia de imágenes de forma real y cronológica.

Como se puede observar en la Figura 26 las primeras imágenes del vídeo son decodificadas a partir de 2 GOP (GOPs abiertos) tras lo que se decodifican las capas en el orden indicado por las flechas.

Para la **escalabilidad en calidad** (Figura 27) las sub-bandas temporales deben ser decodificadas usando el orden **LRCP**.

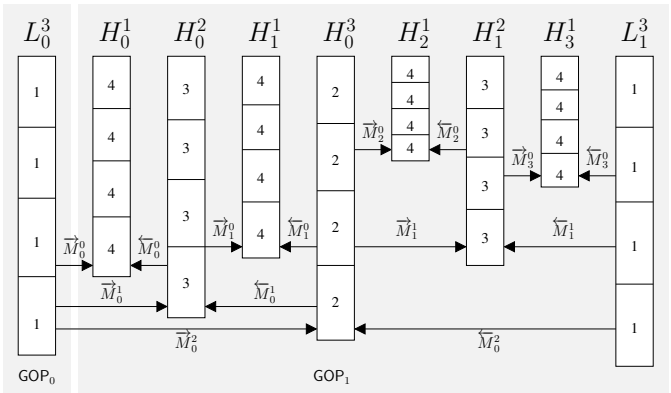

Figura 26. Escalabilidad temporal en MCJ2K.

Éste orden se organiza por capas de calidad, resolución, componentes y finalmente precintos. Es el tipo de orden que guarda el mejor compromiso entre la resolución y la calidad en la imagen que realmente es visualizada. De manera, se cuida que no se de el caso de ver una imagen demasiado pequeña con mucha calidad, o demasiado grande con una calidad muy deficiente. Recordemos que realmente la calidad de una imagen consiste en la cantidad de altas frecuencias que transmitimos de ella. A mayor cantidad de éstas, mas detalle tiene el vídeo y, a menor cantidad, se presenta mas suavizado, como desenfocado. En la practica se suele dejar que el propio ancho de banda disponible trunque este número de paquetes.

Se muestra la misma gráfica para la **escalabilidad en calidad y espacial** porque, ambas necesitan la misma ordenación de datos. Eso quiere decir que, si deseamos una reconstrucción por niveles de resolución lo que hay que hacer es usar el orden que genera la escalabilidad en calidad y descartar aquellos datos que pertenecen a subbandas DWT que generan una imagen de mayor resolución que la que se esta reconstruyendo en ese momento. El orden usado se denomina **RLCP** de las siglas en inglés de *Resolution Layer Component Precint* este tipo de ordenación consiste en que el primer paquete tiene

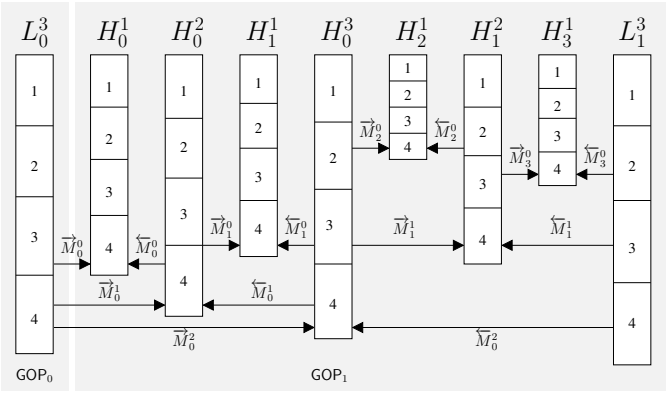

Figura 27. Escalabilidad espacial y en calidad de MCJ2K.

toda la imagen a baja resolución, el segundo y sucesivos continen información que amplia la resolución, hasta llegar a la máxima.

III-K4. Organización de stream: Para concretar la sintaxis de las Figuras 26 y 27 observemos que cada sub-banda temporal $(L^t, H^t, H^{t-1}, \cdots, H^1)$ es un stream diferente, comprimido con Motion JPEG 2000 de forma independiente.

Aunque se conoce el número de imágenes en cada sub-banda, cada imagen puede tener distinta peso (Kb). Por lo tanto, con el fin de decodificar sólo una imagen dada, es necesario descomprimir diferentes sub-bandas.

IV. Arquitectura cliente/servidor para el streaming de vídeo

IV-A. La arquitectura cliente/servidor

La arquitectura cliente/servidor (en adelante, arquitectura C/S) es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, que le da respuesta.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La arquitectura C/S sustituye a la arquitectura monolítica en la que no hay distribución, tanto a nivel físico como a nivel lógico. En ella, el remitente de una solicitud es conocido como cliente. Sus características más importantes podrían resumirse en que:

- Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación (dispositivo maestro o amo).
- Espera y recibe las respuestas del servidor.
- Por lo general, puede conectarse a varios servidores a la vez.
- Normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.
- El cliente presenta una velociad de conexión con el servidor, que estará limitada por la contratación de red que haya echo el usuario con una compañía y por las limitaciones propias del enlace. Por ejemplo: Si el cliente ha contratado un servicio de 6 Mbps este será la capacidad máxima del cliente para recibir datos, además aunque se le proporcionara un ancho de banda superior no se podría alcanzar una velocidad mayor de 10Mb si el cable utilizado para el enlace es, por ejemplo, cable coaxial.

Por otra parte, al receptor de la solicitud enviada por el cliente se conoce como servidor. Sus principales características son:

- Al iniciarse esperan a que lleguen las solicitudes de los clientes. Desempeñan entonces un papel pasivo en la comunicación (dispositivo esclavo).
- Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
- Por lo general, aceptan conexiones desde un gran número de clientes (aunque en ciertos casos el número máximo de peticiones puede estar limitado por motivos de rendimiento y seguridad).
- No es frecuente que interactúen directamente con los usuarios finales (sin un cliente típico).

Como es lógico la utilización de este tipo de arquitectura presenta una serie de ventajas y de inconvenientes. Entre las primeras destacaríamos:

- **Centralización del control:** los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema. Esta centralización también facilita la tarea de poner al día datos u otros recursos.
  La centralización del sistemas tiene la cualidad de permitir un control más sencillo, ya que es la mejor forma de captar, manipular y usar la información cuando es necesario que un gran número de usuarios puedan acceder a ella. Otra de sus ventajas es que evita la inconsistencia de las aplicaciones. De este modo, se obtiene una forma sencilla y que requiere poco recursos, para controlar un sistema.
- **Cierto grado de escalabilidad:** se puede aumentar la capacidad de clientes y servidores por separado. Cualquier elemento puede ser aumentado (o mejorado) en cualquier momento, o se pueden añadir nuevos nodos a la red (clientes y/o servidores).
- **Fácil mantenimiento:** al estar distribuidas las funciones y responsabilidades entre varias computadoras independientes, es posible reemplazar, reparar, actualizar, o incluso trasladar un servidor, mientras que sus clientes no se verán afectados por ese cambio (o se afectarán mínimamente). Por lo tanto, podríamos decir que hay una independencia de los cambios.
- **Se trata de una arquitectura muy desarrollada y extensivamente probada:** existen tecnologías, suficientemente desarrolladas, diseñadas para el paradigma de C/S que aseguran la seguridad en las transacciones, la

amigabilidad de la interfaz, y la facilidad de empleo.

Por otra parte, es posible encontrar algunas desventajas, que son comentadas a continuación:

- **Existencia de cuellos de botella:** la congestión del tráfico ha sido siempre un problema en el paradigma de C/S. Cuando una gran cantidad de clientes envían peticiones simultaneas al mismo servidor, puede ser que cause muchos problemas para éste (a mayor número de clientes, más problemas para el servidor en relación a consumo de ancho de banda, memoria y CPU). De ahí, que surgieran los sistemas descentralizados.
- **Débil tolerancia a fallos del servidor:** cuando un servidor está caído, las peticiones de los clientes no pueden ser satisfechas. Este problema se soluciona, teniendo un grupo de servidores en lugar de uno solo.
- **Altos recursos computacionales y de ancho de banda en el lado servidor:** el software y el hardware de un servidor son generalmente muy determinantes. Un hardware regular de un ordenador personal puede no poder servir a cierta cantidad de clientes. Normalmente se necesita software y hardware específico, sobre todo en el lado del servidor, para satisfacer el trabajo. Por supuesto, esto aumentará el coste.
- **Efectos colaterales causados por la seguridad:** el cliente no dispone de los recursos que puedan existir en el servidor. Por ejemplo, si la aplicación es una página Web, no podemos escribir en el disco duro del cliente o imprimir directamente sobre las impresoras sin sacar antes la ventana previa de impresión de los navegadores.

La mayoría de los servicios de Internet son tipo de cliente-servidor. La acción de visitar un sitio Web requiere una arquitectura cliente-servidor, ya que el servidor web sirve las páginas Web al navegador (al cliente). Por ejemplo, al leer un artículo del portal de un periódico de Internet, la computadora y el navegador Web del usuario serían considerados un cliente; y las computadoras, las bases de datos, y los usos que componen el portal serían considerados el servidor. Cuando el navegador Web del usuario solicita un artículo particular del periódico, el servidor Web recopila toda la información a mostrar en la base de datos, la articula en una página Web, y la envía de nuevo al navegador Web del cliente.

IV-B. Hilos, servidores concurrentes e iterativos

Un servidor concurrente atiende a varios clientes al mismo tiempo, más aún, mientras está atendiendo sigue escuchando. El problema es que todo cliente tiene que esperar su turno para ser atendido, si uno de ellos pide un archivo muy grande los demás tienen que esperar. Por lo tanto, podríamos decir que el procedimiento que sigue el servidor es el siguiente:

1. Espera por la llegada de un cliente.
2. Inicia un servidor para manejar los requerimientos del cliente. Esto involucra la creación de un nuevo proceso o hilo. Cuando el cliente se va (termina) el proceso o hilo también termina.
3. Regresa al primer paso.

En el anterior esquema entendemos que un hilo es un flujo de control independiente sobre una sección de código dentro de un proceso que puede ser planificado por el sistema operativo para ejecutarse. Existen dentro de un proceso y para que puedan ser planificadas deben mantener información sobre el flujo de control: puntero a la pila, registros, conjunto de señales pendientes y bloqueadas, propiedades de planificación (como la política y la prioridad) y otros datos específicos de la hebra.

IV-C. La pila de protocolos TCP/IP

Típicamente, el protocolo utilizado en las aplicaciones cliente/servidor el es TCP/IP. Se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), que fueron dos de los primeros en definirse, y que son los más utilizados de la familia.

En la pila de protocolos TCP/IP, TCP es la capa intermedia entre el protocolo de Internet (IP) y la aplicación. Habitualmente, las aplicaciones necesitan que la comunicación sea fiable y, dado que la capa IP aporta un servicio de datagramas no fiable (sin confirmación), TCP añade las funciones necesarias para prestar un servicio que permita que la comunicación entre dos sistemas se efectúe libre de errores, sin pérdidas y con seguridad.

Los servicios provistos por TCP corren en el anfitrión (host) de cualquiera de los extremos de una conexión, no en la red. Por lo tanto, TCP es un protocolo para manejar conexiones de extremo a extremo. Tales conexiones pueden existir a través de una serie de conexiones punto a punto, por lo que estas conexiones extremo-extremo son llamadas en ocasiones, y por simulitud a los circuitos creados en la técnica de conmutación de circuitos, circuitos virtuales.

Las aplicaciones envían flujos de bytes a la capa TCP para ser enviados a la red. TCP divide el flujo de bytes llegado de la aplicación en segmentos de tamaño apropiado (normalmente esta limitación viene impuesta por la unidad máxima de transferencia o MTU (Maximum Transfer Unit) del nivel de enlace de datos de la red a la que la entidad está asociada) y le añade sus cabeceras. Entonces, TCP pasa el segmento resultante a la capa IP, donde a través de la red, llega a la capa TCP de la entidad destino. El TCP comprueba que ningún segmento se ha perdido asignando a cada uno un número de secuencia, que es también usado para asegurarse de que los paquetes han llegado a la entidad destino en el orden correcto. El TCP devuelve un asentimiento (ACK) por bytes que han sido recibidos correctamente; un temporizador en la entidad origen del envío causará un timeout si el asentimiento no es recibido en un tiempo razonable, y el (presuntamente desaparecido) paquete será entonces retransmitido. TCP revisa que no haya bytes dañados durante el envío usando un checksum; es calculado por el emisor en cada paquete antes de ser enviado, y comprobado por el receptor.

Las conexiones TCP se componen de tres etapas: establecimiento de conexión, transferencia de datos y fin de la conexión. Para establecer la conexión se usa el procedimiento

llamado negociación en tres pasos (3-way handshake). Una negociación en cuatro pasos (4-way handshake) es usada para la desconexión. Durante el establecimiento de la conexión, algunos parámetros como el número de secuencia son configurados para asegurar la entrega ordenada de los datos y la robustez de la comunicación.

Aunque es posible que un par de entidades finales comiencen una conexión entre ellas simultáneamente, normalmente una de ellas abre un socket en un determinado puerto TCP y se queda a la escucha de nuevas conexiones. Es común referirse a ésto como apertura pasiva, y determina el lado servidor de una conexión. El lado cliente de una conexión realiza una apertura activa de un puerto enviando un paquete SYN inicial al servidor como parte de la negociación en tres pasos. En el lado del servidor se comprueba si el puerto está abierto, es decir, si existe algún proceso escuchando en ese puerto. En caso de no estarlo, se envía al cliente un paquete de respuesta con el bit RST activado, lo que significa el rechazo del intento de conexión. En caso de que sí se encuentre abierto el puerto, el lado servidor respondería a la petición SYN válida con un paquete SYN/ACK. Finalmente, el cliente debería responderle al servidor con un ACK, completando así la negociación en tres pasos (SYN, SYN/ACK y ACK) y la fase de establecimiento de conexión.

Durante el establecimiento de conexión TCP, los números iniciales de secuencia son intercambiados entre las dos entidades TCP. Estos números de secuencia son usados para identificar los datos dentro del flujo de bytes, y poder identificar (y contar) los bytes de los datos de la aplicación. Siempre hay un par de números de secuencia incluidos en todo segmento TCP, referidos al número de secuencia y al número de asentimiento. Un emisor TCP se refiere a su propio número de secuencia cuando habla de número de secuencia, mientras que con el número de asentimiento se refiere al número de secuencia del receptor. Para mantener la fiabilidad, un receptor asiente los segmentos TCP indicando que ha recibido una parte del flujo continuo de bytes. Una mejora de TCP, llamada asentimiento selectivo (SACK, Selective Acknowledgement) permite a un receptor TCP asentir los datos que se han recibido de tal forma que el remitente solo retransmita los segmentos de datos que faltan.

A través del uso de números de secuencia y asentimiento, TCP puede pasar los segmentos recibidos en el orden correcto dentro del flujo de bytes a la aplicación receptora. Los números de secuencia son de 32 bits (sin signo), que vuelve a cero tras el siguiente byte después del $2^{32} - 1$. Una de las claves para mantener la robustez y la seguridad de las conexiones TCP es la selección del número inicial de secuencia (ISN, Initial Sequence Number).

Los asentimientos (ACKs o Acknowledgments) de los datos enviados o la falta de ellos, son usados por los emisores para interpretar las condiciones de la red entre el emisor y receptor TCP. Unido a los temporizadores, los emisores y receptores TCP pueden alterar el comportamiento del movimiento de datos. El TCP usa una serie de mecanismos para conseguir un alto rendimiento y evitar la congestión de la red (la idea es enviar tan rápido como el receptor pueda recibir). Estos mecanismos incluyen el uso de ventana deslizante, que controla que el transmisor mande información dentro de los límites del buffer del receptor, y algoritmos de control de flujo, tales como el algoritmo de evitación de la congestión (congestion avoidance), el de comienzo lento (slow-start), el de retransmisión rápidda, el de recuperación rápida (Fast Recovery), y otros.

La fase de finalización de la conexión usa una negociación en cuatro pasos (four-way handshake), terminando la conexión desde cada lado independientemente. Cuando uno de los dos extremos de la conexión desea parar su ”mitad” de conexión transmite un paquete FIN, que el otro interlocutor asentirá con un ACK. Por tanto, una desconexión típica requiere un par de segmentos FIN y ACK desde cada lado de la conexión.

A la luz de toda esta información, se ha elegido trabajar con este protocolo principalmente porque:

1. Ofrece una transferencia fiable de datos (control de errores y de flujo).
2. Realiza el proceso de multiplexación, es decir, el TCP distingue procesos dentro del mismo host a partir del puerto en el que escuchan y a quién están escuchando.
3. Es orientado a conexión (antes de transmitir esta se establece).
4. Aunque las aplicaciones leen y escriben un flujo de bytes, el TCP agrupa los bytes en segmentos que son transmitidos como los datagramas UDP. Por tanto, se trata de un protocolo eficiente desde el punto de vista el overhead de las cabeceras.

IV-D. Código Implementado

El lenguaje de programación que hemos utilizado para implementar este sistema, es el lenguaje C. C es un lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel.

Uno de los objetivos de diseño del lenguaje C es que sólo sean necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que haga falta un soporte intenso en tiempo de ejecución. Es muy posible escribir C a bajo nivel de abstracción; de hecho, C se usó como intermediario entre diferentes lenguajes.

En parte a causa de ser de relativamente bajo nivel y de tener un modesto conjunto de características, se pueden desarrollar compiladores de C fácilmente. En consecuencia, el lenguaje C está disponible en un amplio abanico de plataformas (seguramente más que cualquier otro lenguaje). Además, a pesar de su naturaleza de bajo nivel, el lenguaje se desarrolló para incentivar la programación independiente de la máquina. Un programa escrito cumpliendo los estándares e intentando que sea portátil puede compilarse en muchos computadores.

C se desarrolló originalmente (conjuntamente con el sistema operativo Unix, con el que ha estado asociado mucho tiempo) por programadores para programadores. Sin embargo, ha alcanzado una popularidad enorme, y se ha usado en

contextos muy alejados de la programación de sistemas, para la que se diseñó originalmente.

La implementación de sistemas empleando lenguaje de programación C, tiene una serie de ventajas para el programador:

1. Un núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas.
2. Acceso a memoria de bajo nivel mediante el uso de punteros.
3. Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo.
4. Un sistema de tipos que impide operaciones sin sentido.

Como es lógico, también C presenta una serie de carencias e incovenientes tales como:

1. El mayor problema que presenta el lenguaje C frente a los lenguajes de tipo de dato dinámico es la gran diferencia en velocidad de desarrollo: es más lento programar en C. La razón estriba en que el compilador de C se limita a traducir código sin apenas añadir nada.
2. En C el programador ha de reservar y liberar la memoria explícitamente. En otros lenguajes (como BASIC, Matlab o Java) la memoria es gestionada de forma transparente para el programador.
3. El mantenimiento también es más difícil y costoso que con lenguajes de más alto nivel. El código en C se presta a sentencias cortas y enrevesadas de difícil interpretación.
4. C no dispone de sistemas de control automáticos y la seguridad depende casi exclusivamente de la experiencia del programador.
5. No existe recolección de basura nativa.

Por supuesto, las ventajas de la implementación de código C superan sus inconvenientes, principalmente porque es un lenguaje muy eficiente, puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas. Además, a pesar de su bajo nivel es el lenguaje más portado en existencia y proporciona facilidades para realizar programas modulares y/o utilizar código o bibliotecas existentes.

V. Propuesta

La función del servidor es la de atender a los clientes, proporcionándoles el envío de un determinado vídeo escalable elegido por el usuario. Esta actividad es llevada a cabo mediante la transmisión de diferentes capas de vídeo (niveles de resolución temporal) en función de la capacidad de recibir datos del cliente.

En el presente trabajo el control de flujo de datos no es realizado por el servidor, sino por el cliente que es el que recibir la información, el cuál comprueba en cada slot de tiempo si su ancho de banda es lo suficientemente grande como para recibir una determinada capa de vídeo o por el contrario, su ancho de banda es inferior a lo estipulado y tiene que recibir una capa de vídeo de un nivel más bajo. Un slot de tiempo es, por ejemplo, el tiempo que tarda en consumirse un GOP, que es aproximadamente 1 segundo si tenemos 30 imágenes/segundo y un tamaño de GOP de 32 imágenes.

El control de flujo es posible, gracias a que el cliente posee una serie de buffers de recepción asociadas a cada una de las capas del vídeo y el cliente irá solicitando más o menos capas en función de la velocidad de llenado de esos buffer. Por ejemplo, si se piden 3 capas y los buffers de dichas tres capas comienzan a decrecer, entonces es que el ancho de banda de la red es inferior al bit-rate del vídeo y, por tanto, se debería solicitar una capa menos. Esta velocidad de decrecimiento o de crecimiento se comprueba como varía con respecto al tiempo, por el cliente.

V-A. *Definiciones*

El código del servidor tiene, por defecto, definadas una serie de variables que son propias del protocolo TCP/IP y de los servidores concurrentes:

- **Payload:** se encuentra al inicio del código, antes de que se llegue a ejecutar cualquier procedimiento del servidor. Indica el tamaño del buffer (512 bytes) que sirve de comunicación entre el servidor y el kernel.[4]
- **Puertos:** permite que más de una aplicación utilice la red en un determinado intervalo de tiempo, cada puerto tiene asignado un número entre 0 y 65.535. Los puertos que comprenden desde el 0 al 1.023 están reservados para aplicaciones estándares (como la Web que utiliza el puerto 80). En el caso del servidor, el puerto de escucha se encuentra definido al principio del código por el número 6789.
- **El número máximo de clientes que pueden esperar a que la conexión sea establecida:** esta variable tiene que ver con el consumo de memoria que el kernel dedica a las peticiones de conexión entrantes pendientes de ser procesadas por uno de los hilos del servidor. Al igual que las variables anteriores, esta definida al principio del código, con el valor de 10. Se decidió este valor porque era lo suficientemente alto para comprobar la concurrencia del servidor sin que por ello, se necesitase demasiados recursos por parte de la máquina que se encargara de ejecutar el código del servidor.

V-B. *Funcionamiento del Servidor*

A continuación se muestra y describe, el funcionamiento de las partes del servidor mas relevantes, junto con el código en C asociado:

1. Creación del socket de escucha:

```
1  int listen_sd; {
2
3    /* Obtenemos el número del protocolo. */
4    struct protoent *ptrp; /* Puntero a un ↩
         protocolo de entrada de la tabla. */
5    ptrp = getprotobyname("tcp");
6    if(ptrp ==NULL) {
```

[4]Lo que no significa que el tamaño de los payloads de todos los segmentos que envía el servidor tenga que ser de 512 bytes. El TCP regula dicho tamaño en función del control de flujo que realiza el cliente, del MTU y de la congestión de la red.

```
      perror("getprotobyname");
      exit(EXIT_FAILURE);
    }

    /* Creamos el socket descriptor. */
    listen_sd = socket (PF_INET, SOCK_STREAM, ↩
        ptrp->p_proto);
    if(listen_sd < 0) {
      perror("socket");
      exit(EXIT_FAILURE);
    }

    /* Usaremos el puerto de servicio sin ↩
        esperar a que éste esté a la espera de ↩
        un time-out de liberación. */
    int yes = 1;
    if(setsockopt(listen_sd, SOL_SOCKET, ↩
        SO_REUSEADDR, &yes, sizeof(int)) < 0) {
      perror("setsockopt");
      exit(EXIT_FAILURE);
    }
}
```

2. Creación del socket de servicio:

```
/* Lazo del servidor. */
  while(1) {

    /* Socket para "servir" a los clientes. */
    int serve_sd;

    /* El servidor se bloquea esperando a que un ↩
        cliente se
       conecte. */ {
                /* Direccion del cliente. */
                  struct sockaddr_in cad;
                  socklen_t alen = sizeof(cad);
                  serve_sd = accept(listen_sd, (↩
                       struct sockaddr *)&cad, &alen↩
                       );
                  if(serve_sd<0) {
                                perror("accept");
                                exit (EXIT_FAILURE);
                  }
        }
```

3. Una vez llega la petición de un cliente, el servidor acepta la petición y asigna un hilo a dicho cliente. También se crean todas las variables necesarias para el proceso de transmisión.

```
/* Hilo que controla a un cliente. */
void *service (void *arg){
/*El socket de comunicacion con el cliente. Notese ↩
    que cada cliente posee una variable "sd" ↩
    diferente. */
int sd = ((int *)arg)[0];
//Indica la cantidad de capas que se le envia el ↩
    servidor al cliente
int contador_capa = ((int *)arg)[1];

/* Asignamos memoria para almacenar los datos ↩
    entrantes */
char *buffer = (char *)malloc(payload_size*sizeof(↩
    char));
int pos_ini, pos_fin;
/* Otras variables necesarias */
int bytes_read, i;
//Se guarda el total de bytes que contiene un buffer↩
    ;
int tamano = 0;
//Cuando se hayan enviado todos los videos se le ↩
    avisa al cliente que deje de recibir
int salir = 0;
pos_ini = 0;
pos_fin = 0;
//Almacena el numero maximo de capas que se pueden ↩
    enviar
int contador_max = contador_capa;

/* Creamos las cadenas con los nombres de los ↩
    archivos */
char fr[] = "frame_types_";
char mr[] = "motion_residue_";
char high[] = "high_";
//Extensiones de los archivos high, low y motion ↩
    residue
char mjc[] = ".mjc";
char nombre[22];
//Se guarda el numero del archivo
int numero = 0;
```

4. Se crean y abren los archivos, teniendo en cuenta que el número de archivos a transmitir varía según el número de capas, por ejemplo, para enviar la capa 1 se necesita solo un archivo, si son dos capas se necesitan 4 archivos, etc...

```
/* Creamos y abrimos los archivos */
FILE *archivo[(contador_capa-1)*3+1];
Vacia_Cadena_Peq(nombre);
sprintf(nombre, "low_%d%s", num_layers-1, mjc);
archivo[0]= fopen(nombre,"rb");
Vacia_Cadena_Peq(nombre);
for (i=0; i<contador_capa-1; i++){
        sprintf(nombre, "%s%d", fr, contador_capa-i↩
            -1);
        numero = 1 + 3*i;
        archivo[numero] = fopen(nombre,"rb");
        if (archivo[numero] == NULL){
                printf ("Ha␣fallado␣el␣archivo␣%s.",↩
                    nombre);
                abort();
        }
        Vacia_Cadena_Peq(nombre);
        sprintf(nombre, "%s%d%s", mr, contador_capa-↩
            i-1, mjc);
        numero = 2 + 3*i;
        archivo[numero] = fopen(nombre, "rb");
        if (archivo[numero] == NULL){
                printf ("Ha␣fallado␣el␣archivo␣%s.",↩
                    nombre);
                abort();
        }
        Vacia_Cadena_Peq(nombre);
        sprintf(nombre, "%s%d%s", high, ↩
            contador_capa-i-1, mjc);
        numero = 3 + 3*i;
        archivo[numero] = fopen(nombre, "rb");
        if (archivo[numero] == NULL){
                printf ("Ha␣fallado␣el␣archivo␣%s.",↩
                    nombre);
                abort();
                }
        Vacia_Cadena_Peq(nombre);
        }
```

5. El servidor envía al cliente un entero con el número máximo de capas de las que se compone el vídeo que va a ser enviado.

```
//Enviamos al cliente el numero de capas maximo del ↩
    que se compone el video que se le va a enviar
send (sd, (void *)&contador_capa, sizeof(↩
    contador_capa), 0);
```

6. El servidor comienza la transmisión de vídeo, enviando el GOP 0. El envío de este GOP es imprescindible para la futura descompresión del GOP 1.

```
/* Comienza la transmision de video enviando el Gop ↩
    0 perteneciente a la primera imagen de Low*/

pos_fin = Leer_Posicion(archivo[0], 1);
fseek (archivo[0], pos_ini, SEEK_SET);
//Calculamos la longitud del archivo
tamano = pos_fin - pos_ini;
//Y se lo enviamos al cliente
send (sd, (void *)&tamano, sizeof(tamano), 0);
fprintf(stderr, "Datos␣Low_%d:␣%d\n", contador_max, ↩
    tamano);
fflush(stderr);
//EnvÃa el GOP 0 al cliente
Clonar_Buffer (buffer, archivo[0], tamano, sd);
```

```
//Recibimos del cliente, un entero que indicara si tiene ancho de banda suficiente
//Si el valor de ese entero es 0, se entrara en el siguiente bucle while
recv (sd, (void *)&contador_capa, sizeof(contador_capa), 0);
//Mientras el cliente siga diciendo que no tiene ancho de banda suficiente para recibir
//El servidor avanzara una posicion en todos los archivos, pero sin enviar nada
// Y el cliente estara insertando imÃ¡genes nulas
while (contador_capa == 0){
  for(i=0; i < contador_max; i++){
        if(i == 0){
        // Archivo Low
        pos_ini = Leer_Posicion(archivo[0], 1);
        fprintf(stderr,"\nNo se ha enviado la capa 0");
        }else{
        //Comienza con Frame_Types
        fseek(archivo[1+3*(i-1)], 1<<(i-1), SEEK_CUR);
        //Comienza con Motion_Residue
        pos_ini = Leer_Posicion(archivo[2+3*(i-1)], 1<<(i-1));
        //Comienza con High
        pos_ini = Leer_Posicion(archivo[3+3*(i-1)], 1<<(i-1));
        fprintf(stderr,"\nNo se ha enviado la capa %d", contador_max-i);
        }
    }
recv (sd, (void *)&contador_capa, sizeof(contador_capa), 0);
}
```

7. Empieza el envío del resto de GOPs, comenzando por el GOP 1. Desde ese instante el cliente le irá indicando al servidor si dispone de suficiente ancho de banda para recibir todas las imágenes, realizándose de este modo una transmisión normal de todas las capas o si por lo contrario, el servidor deberá hacer una gestión de las capas a transmitir.

```
//A partir de aqui se transmiten el resto de GOP, empezando por el GOP 1
for(;;){
/* En el siguiente bucle se controla la gestion de las capas */
for (i=0; i<contador_capa; i++){
    if (i == 0){ //Aqui se envÃa el video low_X
    pos_ini = ftell(archivo[0]);
    pos_fin = Leer_Posicion(archivo[0], 1);
    fseek (archivo[0], pos_ini, SEEK_SET);
    tamano = pos_fin - pos_ini;
    send (sd, (void *)&tamano, sizeof(tamano), 0);
    fprintf(stderr, "Datos Low_ %d: %d\n", contador_max-1, tamano);
    fflush(stderr);
    //Mediante este metodo enviamos el archivo
    Clonar_Buffer (buffer, archivo[0], tamano, sd);
    }else{//Aqui se envian los videos de frame_types, motion_residue y high.
    /*1<<i es igual a 2^i*/
    //Comienza con Frame_Types
    //Se obtiene el total de bytes que componen la cadena de Frame_Types
    tamano = Insertar_Frame (archivo[1 + 3*(i-1)], 1<<(i-1), buffer);
    //Se envia esa cantidad al cliente
    send (sd, (void *)&tamano, sizeof(tamano), 0);
    fprintf(stderr, "Datos Frame_Types_ %d: %d\n", contador_max-i, tamano);
    fflush(stderr);
    //Enviamos el contenido de Frame_Types al cliente
    send (sd, (void *)buffer, tamano, 0);
    //Comienza con Motion_Residue
    pos_ini = ftell(archivo[2 + 3*(i-1)]);
    pos_fin = Leer_Posicion(archivo[2 + 3*(i-1)], 1<<(i-1));
    fseek (archivo[2 + 3*(i-1)], pos_ini, SEEK_SET);
    tamano = pos_fin - pos_ini;
    send (sd, (void *)&tamano, sizeof(tamano), 0);
    fprintf(stderr, "Datos Motion_Residue_ %d: %d\n", contador_max-i, tamano);
    fflush(stderr);
    Clonar_Buffer (buffer, archivo[2 + 3*(i-1)], tamano, sd);
    //Comienza con High
    pos_ini = ftell(archivo[3 + 3*(i-1)]);
    pos_fin = Leer_Posicion(archivo[3 + 3*(i-1)], 1<<(i-1));
    fseek (archivo[3 + 3*(i-1)], pos_ini, SEEK_SET);
    tamano = pos_fin - pos_ini;
    send (sd, (void *)&tamano, sizeof(tamano), 0);
    fprintf(stderr, "Datos High_ %d: %d\n", contador_max-i, tamano);
    fflush(stderr);
    Clonar_Buffer (buffer, archivo[3 + 3*(i-1)], tamano, sd);
    }
}
//En el caso de que no se hayan enviado todas las capas
//debemos leer los videos de las capas restantes ya que ahora esas imagenes no se deben enviar
//y el cliente la rellenarÃ¡ con imÃ¡genes nulas
for(i=contador_capa; i < contador_max; i++){
    if (i == 0){
        pos_ini = Leer_Posicion(archivo[0], 1);
        }else{
        //Comienza con Frame_Types
        fseek(archivo[1+3*(i-1)], 1<<(i-1), SEEK_CUR);
        //Comienza con Motion_Residue
        pos_ini = Leer_Posicion(archivo[2+3*(i-1)], 1<<(i-1));
        //Comienza con High
        pos_ini = Leer_Posicion(archivo[3+3*(i-1)], 1<<(i-1));
        fprintf(stderr,"\nNo se ha enviado la capa %d", contador_max-i);
        }

}
```

8. El servidor recibirá un mensaje del cliente indicando el número de capas que el servidor debería enviar para el siguiente GOP. Se recibe este mensaje para cada GOP, aunque no se haya producido una variación en el ancho de banda del cliente.

```
recv (sd, (void *)&contador_capa, sizeof(contador_capa), 0);
```

9. Después se comprueba si ya se ha llegado al final del archivo, para indicarle al cliente que la tranmisión ha finalizado.

```
//Se establece la condicion en el caso de que se hayan enviado todos los videos
//Obtenemos el byte del archivo de video
getc(archivo[0]);
//Si ese byte era el de final de archivo, se poner la variable salir a 1
//para indicar al cliente que ya puede finalizar el proceso
if (feof(archivo[0])){
   salir = 1;
   send (sd, (void *)&salir, sizeof(salir), 0);
   fprintf(stderr, "Se ha enviado todo.\n");
   fflush(stderr);
    break;
//Si no hemos llegado al final, volvemos a la posicion anterior para seguir enviando
}else{
fseek(archivo[0],-1,SEEK_CUR);
salir = 0;
send (sd, (void *)&salir, sizeof(salir), 0);
 }
}
```

10. Una vez completada la transmisión, se cierran los archivos, finaliza el hilo y conexión con el cliente.

```
//Cerramos los archivos
```

```
2   for (i=0; i<contador_capa; i++){
3     if (i == 0){
4     fclose(archivo[0]);
5     }else{
6     fclose(archivo[1+3*(i-1)]);
7     fclose(archivo[2+3*(i-1)]);
8     fclose(archivo[3+3*(i-1)]);
9      }
10  }
11  fprintf(stderr, "Archivos cerrados!\n");
12  fflush(stderr);
13  /* Cerramos la conexion con el cliente. */
14  close(sd);
15  /* Finalizamos el hilo. */
16  pthread_exit(0);
17  fprintf(stderr, "Finaliza\n"); fflush(stderr);
```

VI. Evaluación

Una vez comprobado que la aplicación funciona correctamente con vídeos de diferentes tamaños, se ha procedido a realizar una serie de pruebas para determinar parámetros de funcionamiento del servidor, tales como el número máximo de clientes que pueden conectarse y el throughput. Además de comprobar que el protocolo funciona según lo previsto, pues esa es la base de este proyecto.

VI-A. Verificación del correcto comportamiento del protocolo

Para poder verificar que la tasa de transmisión de las diferentes capas, por parte del servidor, se ajusta a la capacidad de recepción del cliente, se ha llevado a cabo un experimento que consiste en enviar un vídeo varias veces. Cada envío se hará con una tasa de transmisión diferente, pero para el mismo número de capas y de esto modo, poder comprobar que cuando el cliente no tiene suficiente ancho de banda para recibir datos y le dice al servidor que le transmita menos capas, esta gestión se realiza.

En la elaboración de este experimento se han utilizado dos máquinas virtuales que se encontraban en el mismo ordenador y para poder limitar el ancho de banda se ha utilizado un programa llamado `trickle` [5]. Una vez realizada la transmisión restringida a un número determinado de Kbyte/segundo y descomprimidos los vídeos, se realizó una comparación de los vídeos originales y los obtenidos en la transmisión usando la relación señal/ruido y a partir de esa tabla generamos las siguientes gráficas.

El vídeo que se ha usado se llama `coastguard.yuv`, el cual posee un tamaño de imagen de 352x288 y el tamaño del vídeo es de 45619200 bytes. Antes de realizar el envío, el vídeo ha de ser comprimido con MCJ2K, para ello tenemos que calcular el número total de imágenes que posee el vídeo, porque según las reglas del compresor tenemos que elegir un número de imágenes que sea múltiplo de 32 + 1. Para ello, utilizamos la expresión

$$\frac{\text{Tamaño del vídeo (en bytes)}}{\text{Tamaño de imágen} \times 1.5} \tag{8}$$

En este caso, se multiplica por 1.5 por el formato del vídeo utilizado, que es formato YUV 4:2:2.

[5]Trickle es una veterana utilidad para Linux que nos ayudará a limitar la velocidad de descarga/subida sobre nuestras aplicaciones que deberán ser lanzadas desde terminal.

$$\frac{45619200 \text{ bytes}}{352 \times 288 \times 1.5} = 300 \tag{9}$$

Son 300 imágenes, ahora tomamos un número que sea múltiplo de 32 + 1, pero que no sea mayor de 300. Por ejemplo 289 = 288 + 1 = 32 x 9 + 1. Ya tenemos todos los datos necesarios para comprimir el vídeo y solo quedaría elegir el número de capas y escribir el siguiente comando

```
mcj2k --temporal_levels=5 -pictures=289
```

- **Transmisión del vídeo con una capa:** Como podemos observar en la primera gráfica, para la transmisión de este vídeo necesitamos un ancho de banda superior a 100 y a 600 Kbytes/segundo, aunque obtenemos mejores resultados con 600 Kbytes/segundo puesto que al ser mayor se pueden transmitir más imágenes. No obstante, los resultados son muy deficientes puesto que el cliente pasa la mayor parte del proceso de envío sin recibir datos por falta de ancho de banda.
- **Transmisión del vídeo con 4 capas con diferentes anchos de banda:** A diferencia del caso anterior, el servidor tiene la posibilidad de enviar un número de capas u otro en función del tamaño del ancho de banda del cliente. Los casos más favorables, como es lógico, son los de mayor ancho de banda puesto que el cliente puede recibir más imágenes. Por lo tanto, el peor caso es el de 100 Kbytes/segundo en el que a veces el cliente no puede recibir nada. En cuanto al resto, si bien presentan mejores resultados, ha habido momentos en los que el tamaño de ancho de banda no ha sido suficiente para recibir las 4 capas y el cliente ha tenido que solicitar el envío de capas inferiores de vídeo y cuando si ha sido suficiente ha vuelto a solicitar el envío de las 4 capas.
- **Transmisión de una y 6 capas a 100 KBytes/segundo:** Este caso, como se muestra en la tercera gráfica, es el mejor ejemplo de la eficacia del uso de vídeo escalable en entornos con ancho de banda variable. Lo que se ha hecho es una transmisión, con una tasa de 100 Kbytes/segundo, de una capa y otra transmisión, con la misma restricción de ancho de banda, pero con 6 capas. De tal forma, que para el caso de las 6 capas, si el cliente no tenía suficiente ancho de banda, le solicitaba al servidor el envío de datos de una capa más baja. La diferencia de una transmisión a otra es más que notable, siendo bastante alta la cantidad de imágenes que se han podido recibir en la transmisión de las 6 capas, mientras que por el contrario, con la transmisión de una capa el cliente ha estado más tiempo sin poder recibir imágenes, llegando hasta el punto en el que el cliente ha insertado más imágenes nulas que imágenes de vídeo.

VI-B. Número máximo de clientes que pueden conectarse

Una forma de saber cuál es el número máximo de clientes que pueden conectarse al servidor para ser atendidos en función de las características de la máquina servidora, es dada una máquina con una determinada memoria y CPU, comprobar la necesidad de recursos que tiene el servidor para atender a

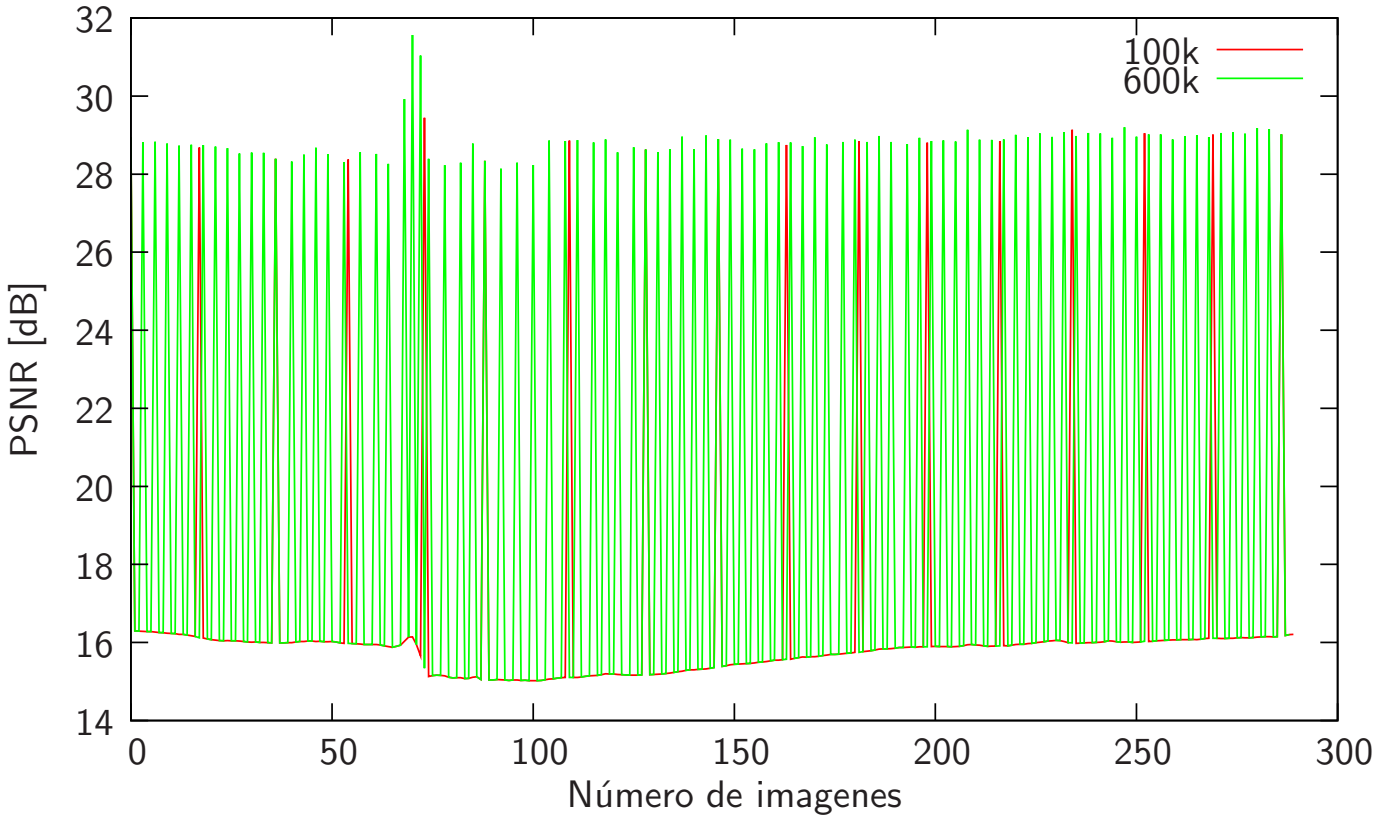

Figura 28. Transmisión del vídeo con una capa

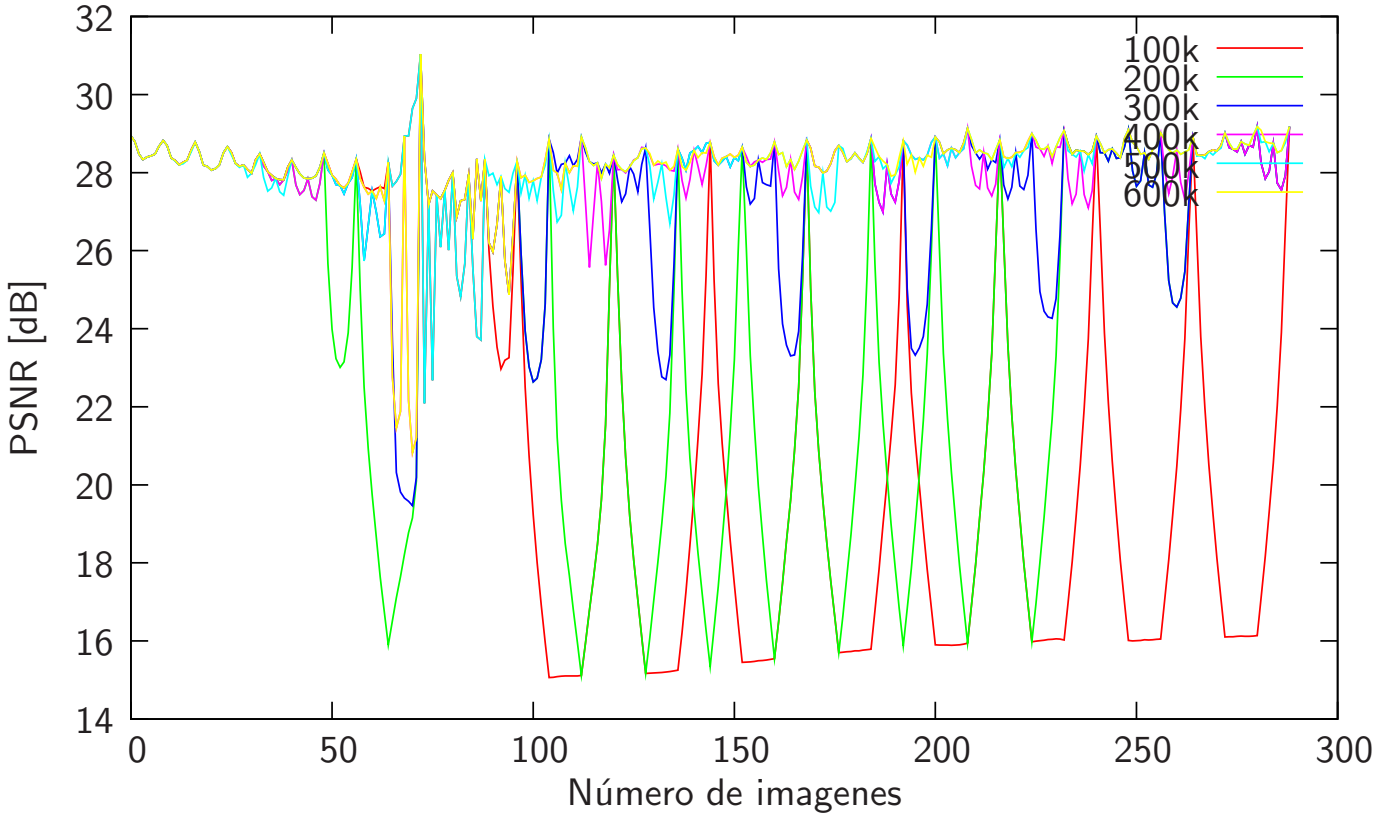

Figura 29. Transmisión del vídeo con 4 capas con diferentes anchos de banda

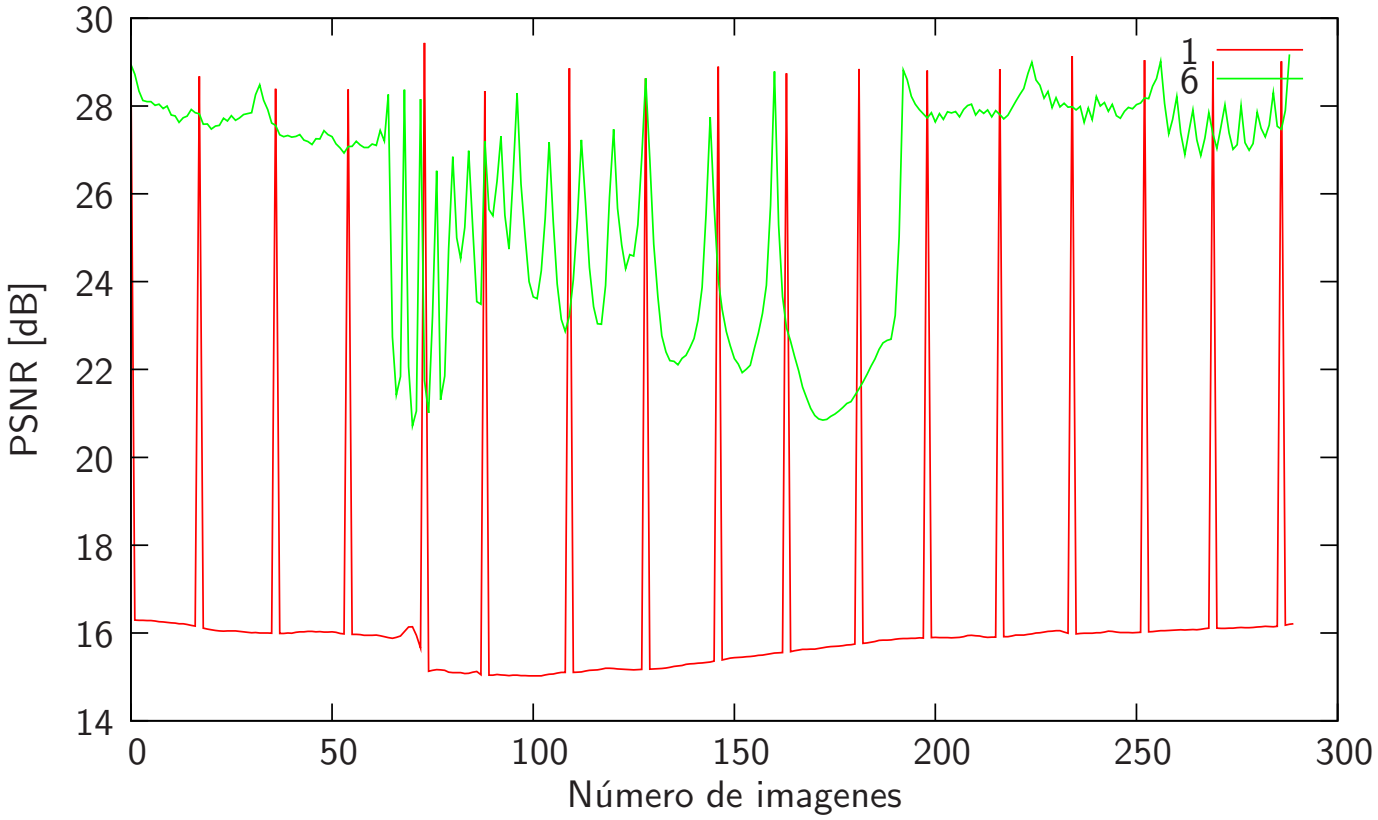

Figura 30. Transmisión de una y 6 capas a 100 Kbytes/segundo.

un cliente y dividir el resultado entre el número de recursos totales.

Por memoria de la máquina servidora nos referimos a la memoria principal, que es donde el computador guarda los datos que está utilizando en el momento presente. El almacenamiento es considerado temporal por que los datos y programas permanecen en ella mientras que la computadora este encendida o no sea reiniciada. Por otra parte, entendemos por CPU, el procesador que es el componente del computador que interpreta las instrucciones contenidas en los programas y procesa los datos.

Para nuestros experimentos disponemos de una máquina donde tenemos instalado el servidor con las siguientes características:

- Tamaño de memoria RAM: 3 GigaBytes.
- Procesador: Intel Core 2 Duo a 1.83 GHz.

Iniciamos la aplicación y un cliente realiza una conexión en la que se le transfiere un vídeo. Mediante el **Administrador de Tareas** comprobamos que el servidor consume de media, desde que acepta la conexión con el cliente hasta que la cierra después de realizar la transmisión, un 3 % de CPU y 0,04 GB (40MB) de de memoria RAM. Dicha medida se ha realizado con diferentes tamaños de vídeo y el consumo ha sido similar. Además, dicho consumo permanece constante a lo largo del periodo de tiempo durante el cual se está realizando el servicio.

Por lo tanto, en la máquina con las características mencionadas anteriormente, nuestro servidor sería capaz de atender de forma simultánea a 33 clientes, teniendo en cuenta que un cliente consume tan solo un 3 %.

VI-C. Throughput

El throughput o número de servicios/segundo de un sistema es el volumen de trabajo o de información que fluye a través de dicho sistema. Para calcular el rendimiento de un servidor de este tipo, lo realizamos de la siguiente forma:

1. En primer lugar, hay que ejecutar el cliente y el servidor en la misma máquina porque si lo hiciéramos desde máquinas diferentes podría ocurrir que las limitaciones del ancho de banda del enlace provocarían un cuello de botella, alterando de este modo el resultado de la prueba de forma que no obtendríamos el verdadero valor de la capacidad del servidor para dar servicios.
2. A continuación suponemos que cada servicio es el envío de un bloque de datos de un vídeo, que como dijimos en el apartado de características del servidor, tiene un valor de 512 bytes.
3. Realizamos el envío de un vídeo que se fragmenta en 1600 bloques de 512 bytes. El tiempo que tarda en realizar la transmisión es, aproximadamente, 0.4 segundos. Por lo tanto, si dividimos el volumen de datos ($1600 * 512 * 8/1024 = 6400$ Kbits) que envía entre el tiempo total de la transmisión obtenemos que el número de Kbps que ofrece el servidor es:

$$\frac{6400 \text{ Kbits}}{0.4 \text{ segundos}} = 16000 \text{ Kbps}. \qquad (10)$$

Como podemos observar el rendimiento del servidor es bastante alto, si lo comparamos con una línea de ADSL doméstica, ya que estaríamos hablando de la posibilidad de enviar datos a una velocidad aproximada de 15.6 Mbps y teniendo en cuenta que normalmente los ISPs ofrecen un ancho de banda de bajada típico de 10Mb, nuestro servidor se adapta más que de sobra al ancho de banda máximo (calidad máxima) que un cliente puede requerir.

Por otra parte, la mayoría de las secuencias de vídeo se reproducen con una calidad más que aceptable si la tasa de bits ronda los 500 Kbps. Por tanto, el número máximo de clientes conectados a esta calidad sería:

$$\frac{16000 \text{ Kbps}}{500 \text{ Kbps/cliente}} = 32 \text{ clientes.} \tag{11}$$

VII. Conclusiones

Con la elaboración de este proyecto hemos conseguido crear un sistema de comunicación Cliente-Servidor basado en la transmisor y recepción de vídeos escalables que han sido codificados mediante MCTF.

Como se ha podido comprobar la eficacia de esta metodología ha sido muy alta porque ha permitido la transmisión de vídeos con anchos de banda muy limitados sin que por ello se viera afectada la calidad de los vídeos. Además el uso de recursos por parte del servidor en la máquina que lo hospeda, es muy bajo por lo que permite atender a un gran número de clientes de forma rápida y eficaz.

Por otro lado, hay que tener en cuenta que para poder visualizar de forma correcta los vídeos de las capas más bajas, se necesita un reproductor especial. Puesto que al tener menos imágenes/segundo el vídeo se reproduce más rápidamente, haciendo imposible su correcta visualización para el ojo humano.

VIII. Trabajos futuros

Si bien se ha conseguido los objetivos planteados inicialmente en este proyecto, que eran la implementación de un servidor de vídeo escalable que adaptase la tasa de transmisión de datos al ancho de banda de recepción del cliente. Siempre todo puede mejorarse y en este caso, sería quitando valores predeterminados del servidor y convertirlos en modificables, tales como: el puerto por el que trabaja, el tamaño de payload, el número máximo de clientes que puede atender de forma simultánea. Y lo que sería aún más importante, incluir la posibilidad de que el servidor pudiera atender a varios clientes que solicitasen vídeos diferentes.

IX. Apéndice

IX-A. Temporización del trabajo realizado

Si tuviéramos que dividir en proyecto de forma temporal, podríamos hacerlo en 5 grandes bloques, atendiendo al tiempo que se le ha dedicado a cada uno de ellos:

1. Búsqueda de antecedentes y planteamiento del proyecto a construir. **Tiempo dedicado: 15 días.**
2. Creación del código de la aplicación Cliente-Servidor. **Tiempo dedicado: 45 días.**
3. Correción de errores para conseguir los objetivos planteados inicialmente. **Tiempo dedicado: 30 días.**
4. Fase de pruebas. **Tiempo dedicado: 60 días.**
5. Elaboración de la memoria del proyecto. **Tiempo dedicado: 60 días.**

Por lo tanto, se podría decir que el tiempo, aproximado, de dedicación a este proyecto ha sido de 7 meses.

IX-B. Código de métodos auxiliares

En la explicación del funcionamiento del servidor, se han empleado una serie de métodos auxiliares que permitían realizar acciones tales como: obtener la posición de una imagen dentro de un archivo, enviar un archivo, insertar caracteres en una cadena, etc. Por ello, a continuación se muestra el código comentado de esta serie de métodos:

- Método para obtener la posición de una imagen dentro de un archivo.

```
int Leer_Posicion(FILE *archivo, int num_imagenes){
  int byte;
  int contador_imagenes = 0;
        int encontrado = 0;
  int i=0;
        int posicion = 0;
//Intentamos abrir el archivo en el que vamos a buscar
        if(!archivo) {
    fprintf(stderr,"Imposible abrir archivo \n");
    abort();
  }
  byte = getc(archivo);
//Recorremos todo el archivo hasta el final
//o hasta que hayamos llegado al lÃmite de las imagenes
  while ((!feof(archivo))&&(contador_imagenes < num_imagenes)){
//Comprueba si el byte es parte de una imagen y si es asi lo copia
    if (byte == 0xff){
      byte = getc(archivo);
                        if (feof(archivo))  break;
//Comprueba si el byte es de la cabecera de una imagen
      if (byte == 0xd9){
                                    encontrado = 1;
      }
    }
                if(encontrado){
                        contador_imagenes++;
                        encontrado = 0;
                }
    if (contador_imagenes < num_imagenes){
                        byte = getc(archivo);
                        if (feof(archivo))  break;
    }
                if (!feof)      printf("\nSe ha llegado al final del archivo.");
  }
//Una vez encontrada la posicion que buscabamos
        posicion = ftell(archivo);
        return posicion;
}
```

- Método para enviar una imagen contenida en un archivo.

```
void Clonar_Buffer(char *buffer, FILE *archivo, int tamano, int sd){
int i, bytes_enviados;
//Se guarda el valor total de bytes,que va a ocupar un paquete a enviar
int tamano_max;
int datos;

while(tamano>0){
```

```
8  //Se comprueba si la longitud de lo que se va enviar↩
       es mayor que la del payload
9    if (tamano > payload_size){
10     tamano_max = payload_size;
11   }else{
12     tamano_max = tamano;
13     Vacia_Cadena(buffer);
14   }
15 //Copiamos todo en el buffer
16   for (i=0;i<tamano_max;i++){
17    buffer[i] = getc(archivo);
18     }
19   datos = 0;
20   while (datos < tamano_max){
21   bytes_enviados = send(sd, (void *)buffer+datos, ↩
         tamano_max-datos, 0);
22   fprintf(stderr, "␣␣␣Bytes␣enviados:␣%d\n", ↩
         bytes_enviados);
23   datos = datos + bytes_enviados;
24   }
25   tamano = tamano - datos;
26   }
27 }
```

- Método que va insertando caracteres de los archivos Frame Types en un buffer.

```
1  int Insertar_Frame (FILE *archivo, int num, char *↩
       cadena){
2  //se le pasa por parametro el archivo FrameType
3  //El numero de caracteres a guardar
4  //Y la cadena donde se guardaran
5    int i;
6
7          if (!archivo){
8                  abort();
9          }
10         Vacia_Cadena (cadena);
11   for (i=0;i<num; i++){
12     cadena[i] = getc(archivo);
13   }
14 //Devuelve la longitud de lo que hemos guardado
15         return i;
16 }
```

BIBLIOGRAFÍA

[1] A. Skodras, C. Christopoulos, and T. Ebrahimi, “”the jpeg 2000 still image compression standard”,” *IEEE Signal Processing Magazine*, vol. 18, no. 5, 2001.

[2] T. J. P. E. G. (JPEG), *Recommendation T.81: Digital Compression and Coding of Continuous-tone Still Images*, International Telecommunication Union (ITU), September 1992.

[3] A. Marcos, *Compresión de imágenes. Norma JPEG*. Editorial Ciencia 3, 1999.

[4] “Motion jpeg,” http://en.wikipedia.org/wiki/Motion_JPEG.

[5] G. K. Wallace, “The JPEG Still Picture Compression Standard,” *Communications of the ACM*, vol. 34, no. 4, pp. 30 – 44, April 1991, se puede conseguir en ftp://ftp.uu.net/graphics/jpeg/wallace.ps.Z.

[6] I. T. 11172-5:1998, “Coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s,” *International Organization for Standardization (ISO)*, 1998.

[7] M. N. and T. D, “Spatial scalability and compression efficiency within a flexible motion compensated 3d-dwt,” *IEEE International Conference on Image Processing (ICIP2004)*, 2004.

[8] T. D., “Successive refinement of video: fundamental issues, past efforts and new directions,” *International Symposium on Visual Communications and Image Processing (VCIP2003), SPIE*, 2003.

[9] L. G. Coloma, *Redes ATM. Principios de interconexión y su aplicación*. Ra-Ma, 2000.

[10] F. Pereira and T. Erahimi, “The mpeg-4 book,” *Pearson Education*, 2002.

[11] I. JTC1/SC29/WG11, “Coding of moving pictures and audio,” *Technical report, International Organisation for Standardisation*, 2002.

[12] F. Pereira and P. Nunes, “Levels for mpeg-4 visual profiles,” *MPEG-4 Industry Forum*, 2002.

[13] W. T. S. G. B. G. L. A., “Overview of the h.264/avc video coding standard,” *Circuits and Systems for Video Technology, IEEE Transactions on*, 2003.

[14] ——, “Overview of the h.264/avc video coding standard,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 2003.

[15] J. R. Ohm, “Three-dimensional subband coding with motion compensation,” *IEEE Transactions on Image Processing*, 1994.

[16] H. S. D. Marpe and T. Wiegand, “Vision general de la extension de vídeo escalable de codificacion de la norma h.264/avc,” *IEEE Transactions on Circuitos y Sistemas de Tecnología de Vídeo*, 2007.

[17] T. J. P. E. Group, *ISO/IEC 15444-1 (JPEG2000, Part 1)*.

[18] D. Taubman, “High performance scalable image compression with ebcot,” *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000.

[19] D. S. Taubman and M. W. Marcellin, “Jpeg2000. image compression fundamentals, standards and practice,” *Kluwer Academic Publishers*, 2002.

[20] F. S. F. G. and J. Mohr, “Motion jpeg2000 for high quality video systems,” *Consumer Electronics, IEEE Transactions on*, 2003.

[21] R. Leung and D. Taubman, “Transform and embedded coding techniques for maximum efficiency and random accessibility in 3d scalable compression,” *IEEE Transactions on Image Processing*, 2004.

[22] L. R. and T. D., “Context modeling and accessibility for 3d scalable compression,” *IEEE International Conference on Image Processing (ICIP2003)*, 2003.

[23] T. J. and T. D., “Optimal erasure protection assignment for scalable compressed data with small packets and short channel codewords,” *EURASIP Journal on Applied Signal Processing; Special issue on Multimedia over IP and Wireless Networks*, 2004.

[24] T. D. and T. J., “Optimal erasure protection for scalably compressed video streams with limited retransmission,” *IEEE Transactions on Image Processing*, 2004.

[25] S. A. and T. D., “Lifting-based invertible motion adaptive transform (limat) framework for highly scalable video compression,” *IEEE Transactions on Image Processing*, 2003.

[26] ——, “Highly scalable video compression using a lifting-based 3d wavelet transform with deformable mesh motion compensation,” *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, 2002.

[27] ——, “Motion-compensated highly scalable video compression using an adaptive 3d wavelet transform based on lifting,” *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, 2001.

[28] G. D. S. M.:, “Multi-frame motion estimation: application to motion compensated prediction,” *Circuits and Systems, 1999. ISCAS ’99. Proceedings of the 1999 IEEE International Symposium on*, 1999.

[29] M. N. and T. D., “A flexible structure for fully scalable motion compensated 3d-dwt with emphasis on the impact of spatial scalability,” *IEEE Transactions on Image Processing*, 2004.

[30] N. Mehrseresht and D. Taubman, “An efficient content-adaptive mc 3d-dwt with enhanced spatial and temporal scalability,” *IEEE Transactions on Image Processing*, 2004.

[31] D. Taubman and A. Secker, “Highly scalable video compression with scalable motion coding,” *IEEE International Conference on Image Processing (ICIP2003)*, 2003.

[32] H. S. D. Marpe; and T. Wiegand, “Overview of the scalable video coding extension of the h.264/avc standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, 2007.

[33] A. Haar, “Zur theorie der orthogolanen funktionen-systeme,” *Mathematische Annalen*, 1910.

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

En este proyecto se ha implementado la parte servidora de un servicio de streaming de vídeo escalable en el tiempo. El vídeo a enviar se ha comprimido previamente mediante MCTF y la tarea del servidor consiste en enviar dicho vídeo a un conjunto de clientes. Lo novedoso de este proyecto es que tanto el servidor como el cliente se comunican, para saber que cantidad de vídeo puede transmitirse en cada instante, en función del ancho de banda disponible, sin que esto provoque interrupciones en su reproducción. Finalmente, mediante una serie de pruebas, se ha comprobado el correcto funcionamiento del protocolo y de la gestión de las diferentes capas de vídeo por parte del servidor.

This project has implemented the server side of a video streaming service scalable in time. The video to send, it has been previously compressed by MCTF and server task is to send the video to a set of clientes. The movelty of this project is that both the server and client commicate, to know how much video can be trasmitted at any instant, depending on available bandwidth, without this leading to interruptions in playback. Finally, through a series of test, we hace found the correct protocol operation and management of the different layers of video from the server.