- **Autor del trabajo:** José Juan Sánchez Hernández.
- **Título:** Transmisión de secuencias de imágenes JPEG 2000 usando actualización condicional controlada por el cliente.
- **Fecha/Convocatoria de defensa:** Septiembre de 2011.
- **Director del trabajo:** Vicente González Ruiz.

Transmisión de secuencias de imágenes JPEG 2000 usando actualización condicional controlada por el cliente

José Juan Sánchez Hernández
Director: Vicente González Ruiz

***Resumen*—Este trabajo propone una estrategia para la visualización remota e interactiva de secuencias de imágenes JPEG 2000 de alta resolución. Las secuencias pueden ser visualizadas en cualquier orden (hacia delante y hacia detrás) y siguiendo cualquier patrón de reproducción y temporización (por ejemplo, parando y continuando la reconstrucción de la secuencia después de un tiempo indefinido y en cualquier parte de ésta). Además por un requerimiento del procedimiento de visualización, el rango de imágenes que forman estas secuencias es típicamente definido en el instante de la visualización, lo que implica que previamente no ha sido posible aplicar ningún proceso de compresión que explote la redundancia temporal. Sin embargo, cierta cantidad de redundancia temporal existe habitualmente en estas secuencias. Para incrementar la calidad de la reconstrucción explotando esta fuente de redundancia, en este trabajo se presenta y se evalúa un algoritmo de actualización condicional de imágenes. Nuestra solución se beneficia de la escalabilidad de tipo SNR y espacial que ofrece el estándar JPEG 2000 para determinar qué regiones de la siguiente imagen deben ser transmitidas y qué otras regiones pueden ser reutilizadas de la imagen previamente reconstruida. Los resultados demuestran que, incluso sin compensación de movimiento y con una transmisión controlada exclusivamente por el cliente, las reconstrucciones son mejores de forma consistente, tanto desde un punto de vista visual como atendiendo a la curva *rate-distortion*, cuando comparamos con aquellas técnicas que sólo eliminan la redundancia espacial. Otras ventajas de nuestra propuesta son: 1) que no se genera un *overhead* de datos durante la transmisión, 2) que el incremento en complejidad computacional es muy pequeño en comparación con técnicas similares y 3) es completamente compatible con cualquier servidor JPIP estándar.**

***Palabras clave*—JPEG 2000, actualización condicional, transmisión de vídeo.**

I. Introducción

Entre las principales características del estándar JPEG 2000 [1] podemos destacar una alta eficiencia en la compresión de imágenes con y sin pérdida, la posibilidad de poder acceder de forma aleatoria al *code-stream* comprimido sin tener que transmitir toda la imagen, la realización de una descompresión progresiva de la imagen y una gran escalabilidad en resolución y calidad. Estas características hacen que JPEG 2000 sea el estado del arte para la exploración de imágenes remotas de gran resolución. Usando el protocolo JPIP (JPEG 2000 *Interactive Protocol*), definido en la Parte 9 [2] del estándar JPEG 2000, los clientes pueden explorar interactivamente imágenes remotas especificando una ventana de interés, a la que llamaremos a partir de ahora en este documento sólo WOI (*Window Of Interest*). Este intercambio de datos entre cliente y servidor, usa todo el ancho de banda disponible de una forma eficiente y no requiere de ninguna codificación o proceso adicional. El servidor extrae solamente los datos requeridos de las imágenes solicitadas y los transmite a sus clientes.

JPEG 2000 ha sido usado satisfactoriamente en muchas áreas científicas como por ejemplo, telemicroscopía o tele-medicina. Una aplicación prometedora en las ciencias del espacio es el proyecto *JHelioviewer* [3], desarrollado por la Agencia Espacial Europea (ESA) [4] en colaboración con la Administración Nacional de Aeronáutica y del Espacio de los Estados Unidos (NASA) [5]. Su principal objetivo es ofrecer una exploración interactiva de las imágenes que son enviadas por el observatorio solar SDO (*Solar Dynamics Observatory*) [6] a la tierra. El observatorio SDO recoge imágenes del sol cada 12 segundos en diez bandas espectrales ultravioletas diferentes con una resolución de 4096×4096 píxeles, lo que quiere decir que envía aproximadamente la cantidad de 1.4 TBytes de imágenes por día.

Desde un punto de vista científico, este elevado volumen de datos plantea varios problemas como pueden ser el acceso a los datos y la exploración

de las imágenes de una forma eficiente. Hoy en día, la combinación del protocolo JPIP y el estándar de compresión JPEG 2000 parece ofrecer la mejor solución para poder examinar de forma eficiente volúmenes de datos de esta magnitud.

La funcionalidad básica de *JHelioviewer* es permitir a los usuarios la exploración de imágenes solares en un determinado instante de tiempo o dentro de un rango de tiempo establecido. Los usuarios pueden solicitar toda la imagen o pueden establecer una WOI, también se pueden mover suavemente a través de una secuencia de imágenes solares, mientras se van visualizando los cambios que producidos en la WOI a lo largo del tiempo del rango establecido.

La comunicación cliente/servidor está basada en el intercambio de peticiones y respuestas. Con cada petición, el cliente especifica, entre otros parámetros, el nombre del archivo remoto que quiere visualizar y una determinada WOI. Los archivos pueden contener una secuencia de N imágenes diferentes (en el caso de *JHelioviewer*, se corresponden a imágenes comprendidas en un determinado rango de tiempo), así que las peticiones deben incluir el rango de imágenes deseado $[a, b]$, con $0 \leq a \leq b \leq N-1$. Sin ninguna interacción extra del usuario, se debe obtener la misma WOI para todas las imágenes comprendidas dentro del rango.

Este trabajo por lo tanto se centra en aplicaciones que hagan uso del protocolo JPIP, como es el caso de *JHelioviewer*, que han sido diseñadas para explorar secuencias de imágenes remotas. La figura 1 muestra un ejemplo de cinco instantes de tiempo durante una sesión de exploración de imágenes remotas usando *JHelioviewer*. Una vez que el usuario ha seleccionado un rango de tiempo, el servidor construye un archivo JPEG 2000 virtual que contiene solamente enlaces a aquellas imágenes cuya fecha y hora pertenecen al rango de tiempo especificado. El cliente inicia una sesión JPIP para este archivo y solicita la primera imagen, que es mostrada en el instante de tiempo t_0. El usuario puede visualizar la secuencia de imágenes que están comprendidas dentro del rango de tiempo, una a una, o saltando a cualquier imagen de la secuencia, y cambiando la dirección en cualquier momento. En este ejemplo, en el instante de tiempo t_2 el usuario hace *zoom* sobre una cierta región de la imagen, cambiando de este modo la WOI actual. Desde esta nueva WOI, el usuario continúa moviéndose hacia delante continuando la secuencia, en los instantes de tiempo t_3 y t_4. En el "modo vídeo", donde las imágenes son visualizadas sin ninguna pausa para un *frame-rate* que el usuario puede controlar, la cantidad de datos que se obtienen del servidor JPIP depende del ancho de banda disponible y del *frame-rate* establecido. En cualquier caso, las imágenes son transmitidas siguiendo una progresión en orden de calidad de modo que se va incrementando la calidad de las reconstrucciones al mismo tiempo que se van recibiendo los datos del servidor.

El hecho de que el observatorio SDO tome imágenes del sol a una alta frecuencia temporal, provoca que exista una gran correlación espacial entre las imágenes consecutivas, alrededor de 33dB en términos de PSNR (*Peak Signal-to-Noise Ratio*) como media entre imágenes[1], y por lo tanto, un alto grado de redundancia temporal. Por esta razón, hemos centrado este trabajo en la eliminación de esta redundancia haciendo uso de la técnica de actualización condicional.

II. Arquitectura JHelioviewer

La figura 2 muestra un diagrama con la arquitectura de la aplicación *JHelioviewer*. *JHelioviewer* está basado en una arquitectura cliente/servidor, donde la comunicación que se realiza entre el *browser* de la parte cliente y el servidor JPIP está basada en mensajes de petición y respuesta usando el protocolo JPIP, que viaja encapsulado dentro del protocolo HTTP. Las imágenes JPEG 2000 están almacenadas en un repositorio de imágenes, mientras que los metadatos de las imágenes se encuentran en otro repositorio diferente. Este repositorio contiene información de interés sobre las observaciones realizadas en las imágenes JPEG 2000. El repositorio con los metadatos de las imágenes puede estar integrado directamente en el servidor de imágenes o puede interactuar como un elemento independiente.

Cada elemento de la arquitectura *JHelioviewer* incluye a su vez varios elementos. De este modo, el *browser* incluye una interfaz de usuario, un decodificador de imágenes, un gestor de capas y eventos,

[1]Este valor ha sido determinado haciendo la media del PSNR de una imagen respecto a la siguiente, en una secuencia de imágenes tomadas con una cadencia de 432 segundos. Sin embargo, el observatorio SDO ofrece la posibilidad de obtener imágenes con una mayor cadencia, aproximadamente cada 12 segundos lo que implicaría aún mayores índices de correlación temporal.

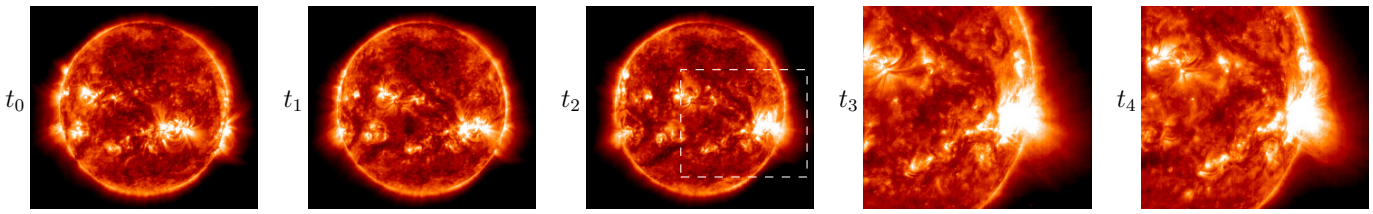


Figura 1. Un ejemplo donde se muestran diferentes instantes de tiempo de una sesión de exploración de imágenes remotas.

un manejador de metadatos y una caché local de imágenes. El servidor contiene un *parser* JPEG 2000 y un repositorio de imágenes JPEG 2000.

A continuación se detalla la funcionalidad de cada uno de los elementos que interactúan en el *browser* del lado del cliente:

- La **interfaz de usuario** es la encargada de mostrar las imágenes y las animaciones. También permite realizar algunas operaciones básicas de procesamiento de imágenes.
- La **caché local de imágenes** contiene una caché con los datos que han sido transmitidos por el servidor previamente.
- El **gestor de capas** permite al usuario la posibilidad superponer capas con diferentes imágenes JPEG 2000.
- El **gestor de eventos solares** interactúa internamente con el gestor de capas y externamente con el repositorio de eventos a través de un *Web Service*.
- El **renderizador de imagen** recibe el *stream* JPIP con las cabeceras de la imagen y renderiza la petición del usuario a la resolución solicitada.

Los elementos del lado del servidor incluyen las siguientes funcionalidades:

- El ***parser*** recibe una petición de una imagen con una serie de parámetros (como por ejemplo: tamaño de la ventana, *offset*, capas de calidad o niveles de resolución). Éste consulta el repositorio de imágenes y devuelve una respuesta al cliente con los datos solicitados en forma de *stream* JPIP.
- El **repositorio de imágenes** es un sistema de archivos que almacena imágenes comprimidas en JPEG 2000 usando una estructura de directorio jerárquico. Los metadatos asociados a cada una de las imágenes (como por ejemplo: fecha/hora de la observación, instrumento, detector y observatorio) son almacenados en el catálogo de metadatos.
- El **modelo de caché del cliente** contiene información sobre la caché local de cada cliente. El *parser* utiliza esta información para evitar volver a enviar información que ya ha sido enviada al cliente y ya se encuentra almacenada en la caché del cliente.

El catálogo de metadatos está almacenado en una base de datos relacional y es accesible a través de un *Web Service*. El motivo de separar este catalogo de metadatos del servidor JPIP fue para evitar que el rendimiento del servidor se viese afectado por el funcionamiento de éste.

El catálogo de eventos solares permite a los usuarios consultar y visualizar eventos solares de cierta relevancia.

III. El estándar JPEG 2000

JPEG 2000 es un estándar de compresión y codificación digital de imágenes, que fue creado por el grupo JPEG (*Joint Photographic Experts Group*). En Marzo de 1997 los expertos en fotografía del grupo JPEG propusieron desarrollar un nuevo sistema de compresión de imágenes digitales, con la intención de sustituir el formato original, cuyo nombre está formado por las iniciales del grupo JPEG. Este nuevo sistema de compresión debía resolver las carencias que presentaba el actual formato JPEG, utilizar la Transformada Discreta *Wavelet* o DWT (*Discrete Wavelet Transform*) y cubrir las nuevas necesidades que iban apareciendo con el avance de las nuevas tecnologías.

Después de más de dos años evaluando todas las propuestas recibidas, una versión modificada del algoritmo EBCOT (*Embedded Block Coding with Optimized Truncation*), desarrollado por David Taubman, fue finalmente la propuesta elegida como el núcleo del nuevo estándar de compresión. Este

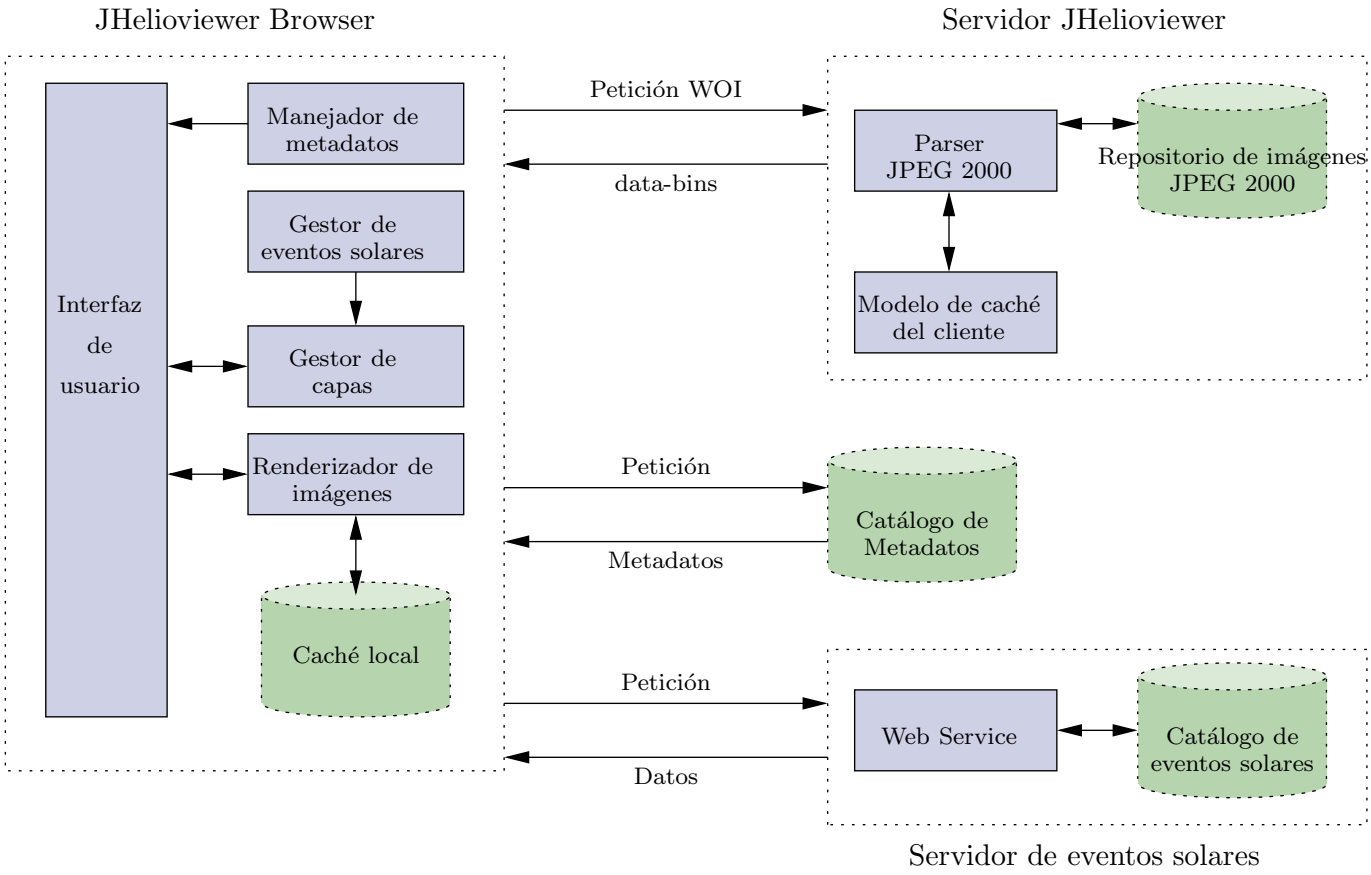


Figura 2. Arquitectura JHelioviewer. La arquitectura incluye tres partes básicas, el *browser* (cliente), el servidor JPIP y el servidor de eventos solares. El catálogo de metadatos se puede considerar que es un elemento que forma parte del servidor JPIP

núcleo fue complementado con la definición de un nuevo formato de archivo, y una forma eficiente de particionar los datos con el objetivo de sacar el máximo partido a la escalabilidad que ofrece el algoritmo EBCOT. Todo esto fue la base del nuevo estándar, que fue llamado JPEG 2000. Esta primera parte son los requisitos mínimos requeridos por cualquier implementación de este estándar.

Desde un punto de vista formal, podemos decir que JPEG 2000 está dividido en 11 partes diferentes:

- **Parte 1**: Describe los mínimos requisitos que debe tener un codificador JPEG 2000. El objetivo de esta parte es crear un conjunto de operaciones básicas con el mayor grado de interoperabilidad entre las diferentes implementaciones del estándar.
- **Parte 2**: Contiene extensiones opcionales que mejoran el rendimiento en algunos campos, del sistema básico de codificación.
- **Parte 3**: Hace referencia a "*Motion* JPEG 2000", un estándar para el vídeo digital de alta calidad.
- **Parte 4**: Es un conjunto de pruebas de verificación que tienen el fin de certificar los procesos de codificación y decodificación del estándar.
- **Parte 5**: Conjunto de modelos *software* que deben ser usados como referencia en el desarrollo de productos JPEG 2000.
- **Parte 6**: Su objetivo es establecer un conjunto de especificaciones para el uso de JPEG 2000 en diferentes aplicaciones como puede ser el *fax*, publicaciones electrónicas y aplicaciones numéricas.
- **Parte 7**: Esta parte ha sido abandonada.
- **Parte 8**: JPSEC (JPEG 2000 *Secure*). Esta parte se ocupa de aspectos de seguridad para JPEG 2000 como puede ser la encriptación, las marcas de agua, etc.
- **Parte 9**: JPIP, protocolos interactivos y APIs (*Application Protocol Interfaces*). Esta parte

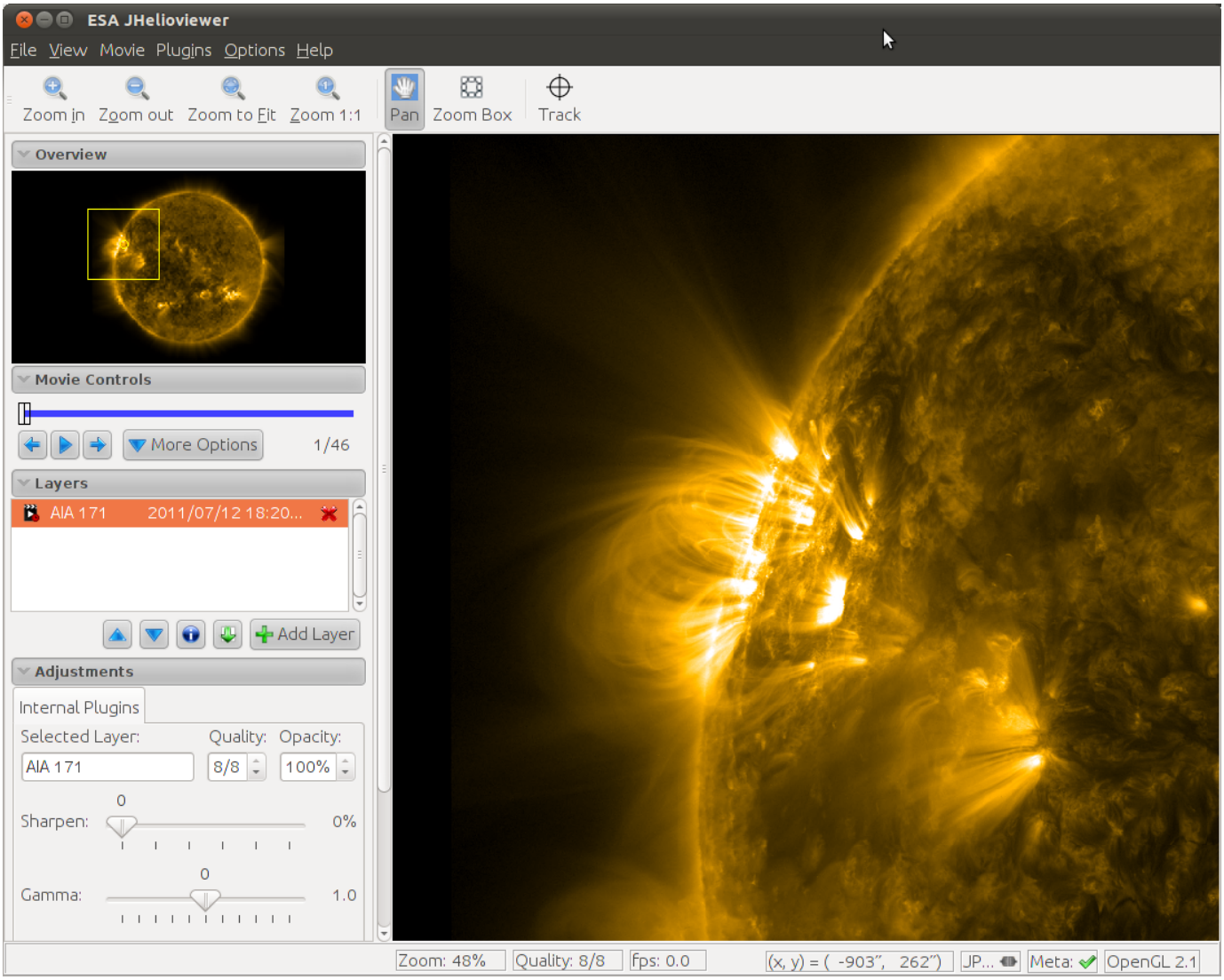


Figura 3. *Screenshot* de la aplicación JHelioviewer. A la izquierda-centro de la imagen se puede ver el gestor de capas y los controles del modo vídeo.

define un protocolo interactivo de red y las herramientas específicas para realizar el intercambio de imágenes JPEG 2000 y sus metadatos, de forma eficiente.

- **Parte 10**: JP3D, imágenes volumétricas. Esta parte desarrolla aspectos como la reconstrucción de imágenes médicas en 3D.
- **Parte 11**: *Wireless* JPWL. Esta parte hace referencia a las aplicaciones *wireless* multimedia. Su objetivo es la detección y corrección de errores JPEG 2000 en un entorno propenso a errores.

III-A. Características

Algunas de las características que hacen que este estándar sea el ideal para la codificación y transmisión de imágenes son las siguientes:

- Ofrece altos ratios de compresión, en términos de calidad visual y PSNR (*Peak Signal-to-Noise Ratio*) a *bit-rates* muy pequeños. Incluso por debajo de 0.25 bits/píxel ofrece mejores resultados que el algoritmo JPEG *Baseline*.
- Es muy robusto ante la presencia de errores y tiene capacidad de recuperación ante los mismos.
- Permite realizar compresión sin pérdidas y con pérdidas.
- Permite acceder de forma aleatoria al *code-*

stream. De este modo es posible manipular ciertas zonas de la imagen o regiones de interés.

- Ofrece escalabilidad por resolución y calidad. Es posible organizar el *code-stream* de modo que las reconstrucciones de la imagen vayan incrementando su calidad a medida que se va recibiendo mayor número de *bits*. Esto es posible debido una decodificación progresiva por planos de bits, donde vamos decodificando desde el plano de *bits* más significativo hasta el menos significativo. El *code-stream* también se puede organizar por niveles de resolución, de modo que vayamos incrementando el nivel de resolución de las reconstrucciones de la imagen a medida que recibimos mayor número de *bits*.
- Permite la codificación de regiones de interés (ROI) dentro de la imagen. El usuario puede seleccionar determinadas regiones de la imagen para que sean codificadas con una mayor precisión que el resto de la imagen. Por ejemplo, un médico puede encontrar una cierta región (o varias regiones) de una radiografía digital más interesante que otras regiones. Por lo tanto, es posible comprimir la radiografía digital de tal manera que la región de interés se comprima sin pérdidas y el resto de la imagen con pérdidas, de modo que podamos almacenarla con el tamaño de archivo deseado.
- El estándar JPEG 2000 permite al usuario seleccionar el tamaño que desea que tenga el archivo comprimido. Esto es posible porque el proceso de compresión se realiza por planos de *bits*, de modo que el proceso es interrumpido cuando se alcanza el tamaño deseado.
- Dispone de un formato de archivo muy sencillo y flexible.

De todas las características comentadas anteriormente, la alta escalabilidad por resolución y calidad son quizás las más atractivas de todas. La escalabilidad por resolución permite al servidor extraer solamente los datos necesarios para reconstruir una WOI específica, sin ningún procesamiento adicional. Gracias a la transformada DWT las imágenes son almacenadas con una representación con múltiples niveles de resolución espacial, de modo que un cliente puede obtener una WOI, a un bajo nivel de resolución sin ningún procesamiento adicional. La escalabilidad en calidad permite al usuario visualizar las reconstrucciones de las regiones solicitadas con una calidad que va incrementándose a medida que se van recibiendo más datos del servidor. Hoy en día no existe otro estándar que ofrezca estas posibilidades.

III-B. Particiones de los datos

El estándar JPEG 2000 define una amplia variedad de particiones para los datos de la imagen, con el objetivo de explotar al máximo la escalabilidad ofrecida. Todos estos tipos de particiones permiten una manipulación eficiente de la imagen, o de parte de ésta. La figura 4 muestra un ejemplo gráfico de las principales particiones que se pueden realizar.

Con el objetivo de comprender el concepto de cada partición definida en el estándar JPEG 2000, es necesario aclarar el concepto de *canvas*. El *canvas* es una zona de dos dimensiones donde se mapean todas las particiones relacionadas con la imagen. A partir de ahora en adelante, todo el sistema de coordenadas utilizado está relacionado con el *canvas*, cuyo tamaño, ancho (I_2) y alto (I_1), se corresponden con el tamaño total de la imagen asociada. Cada partición está situada y mapeada sobre el *canvas* de una forma específica. Una imagen puede estar compuesta por una o más componentes. En la mayoría de los casos las imágenes solamente tienen tres componentes: rojo, verde y azul (RGB), con un tamaño igual al tamaño del *canvas*.

El estándar JPEG 2000 permite dividir una imagen en pequeñas regiones rectangulares llamadas *tiles*. Cada *tile* está comprimido de forma independiente respecto al resto, por lo tanto los parámetros utilizados durante el proceso de compresión pueden ser diferentes para los distintos *tiles*. Por defecto siempre se utiliza un único *tile*, que equivale a toda la imagen.

Otra de las posibles aplicaciones del uso de las particiones en *tiles* es con imágenes que contienen diferentes elementos y están visualmente separados, como puede ser texto, gráficos o materiales fotográficos. Cuando esto no sea así, y las imágenes sean continuas y homogéneas, no es recomendado el uso de *tiles* porque éste produce artefactos en los bordes de los *tiles*, causando el efecto de mosaico. Además, el tamaño de imagen comprimida es mucho mayor si se usan *tiles*.

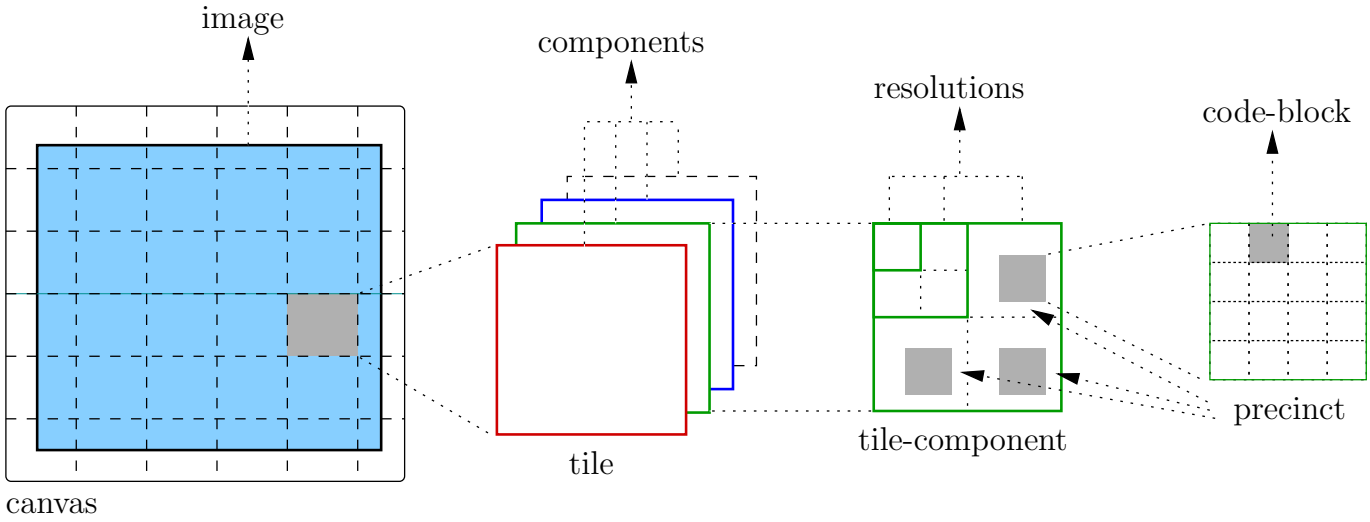


Figura 4. Particiones de los datos definidas por el estándar JPEG 2000.

La transformada DWT y todas las etapas de cuantificación/codificación son aplicadas de forma independiente a cada uno de los *tile-component*. Un *tile-component*, de un *tile* t y una componente c, está definido por una zona bidimensional limitada por t tomando en cuenta la zona ocupada por c. Esto quiere decir que, si una imagen tiene solamente un *tile*, con tres componentes de color, hay tres *tile-components*, que son comprimidas de forma independiente.

Para cada *tile-component*, identificado por el *tile* t y la componente c, hay un total de $D_{t,c} + 1$ resoluciones, donde $D_{t,c}$ es el número de etapas aplicadas en la DWT. El r-ésimo nivel de resolución de un *tile-component* comprimido se obtiene después de aplicar r veces la transformada inversa DWT. El valor de r está comprendido en el rango $0 \leq r \leq D_{t,c}$.

Cada *tile-component*, después de aplicar la transformada DWT, es dividido en *code-blocks*, que son codificados independientemente. En cada nivel de resolución r de cada *tile-component* (t, c), los *code-blocks* son agrupados en *precincts*. Esta partición está definida por el alto, $P_1^{t,c,r}$, y el ancho, $P_2^{t,c,r}$, de cada *precinct*. El número de *precincts* en vertical, $N_1^{P,t,c,r}$, como en horizontal, $N_2^{P,t,c,r}$ viene dado por la siguiente expresión:

$$N_i^{P,t,c,r} = \left\lceil \frac{I_i}{2^{D_{t,c}-r} P_i^{t,c,r}} \right\rceil$$

Los *code-blocks* hacen referencia a los coeficientes *wavelet* generados por la transformada DWT, en el dominio *wavelet*. Sin embargo, los *precincts* hacen referencia a regiones rectangulares en el dominio de la imagen. La escalabilidad espacial ofrecida por el estándar es llevada a cabo por los *precincts*.

El paquete es la unidad fundamental para la organización del *bit-stream* comprimido de una imagen. Cada *precinct* contribuye al *bit-stream* con tantos paquetes como capas de calidad haya. Los datos comprimidos de cada *code-block* están divididos en diferentes segmentos llamados capas de calidad. Todos los *code-blocks* de todos los *precincts* del mismo *tile* están divididos en el mismo número de capas de calidad, además el número de capas de calidad entre *code-blocks* puede ser diferente. Para una cierta capa l, el conjunto de todas las capas l de todos los *code-blocks* relacionados con un mismo *precinct* forman un paquete.

Para decodificar una determinada región de una imagen es necesario decodificar todos los paquetes relacionados con esa región.

Un paquete $\zeta_{t,c,r,p,l}$ está identificado por el *tile* t, la componente c, el nivel de resolución r, el *precinct* p (en coordenadas de *precincts*) y la capa de calidad l.

III-C. Organización del Code-stream

La Parte 1 del estándar JPEG 2000 define la estructura básica para organizar los datos comprimidos de una imagen en *code-streams*. Un *code-stream* incluye todos los paquetes por el proceso de

compresión de una imagen más un conjunto de marcadores, que se utilizan para señalar ciertas partes, así como para incluir la información necesaria para realizar la descompresión.

El *code-stream* es un sencillo formato de archivo para una imagen JPEG 2000. Cualquier descompresor estándar debería entender un *code-stream* almacenado en un archivo. A este formato básico de archivo también se le conoce como formato *raw*, y se suele utilizar con la extensión ".J2C".

Los marcadores tienen un único identificador, que consiste en un entero sin signo de 16 *bits*. Estos marcadores pueden aparecer solos, o acompañados de información adicional. Cuando un marcador va acompañado de información adicional se le conoce como marcador de segmento.

Los marcadores de segmento tienen, después del identificador, otro entero sin signo de 16 *bits* indicando el número de *bytes* de los datos que acompañan al marcador, incluyendo también los dos *bytes* de este entero, pero sin contar los dos *bytes* del identificador.

El *code-stream* siempre empieza con el marcador SOC (*Start Of Code-stream*), el cual no contiene ninguna información adicional. Después de este marcador aparecen un conjunto de marcadores conocidos como cabecera principal. Después del marcador SOC siempre hay un marcador SIZ (*Image and tile Size*), con la información global necesaria para poder descomprimir los datos, por ejemplo: el tamaño de la imagen, el tamaño de los *tiles*, el punto de anclaje de los *tiles*, el número de componentes, los factores de sub-muestreo, etc.

Hay otros dos marcadores que deben aparecer de forma obligatoria en la cabecera: COD (*Coding style Default*), con la información relacionada a la codificación de la imagen, como el número de capas de calidad, el número de etapas DWT, el tamaño de los *code-blocks*, el tipo de progresión, etc.; y QCD (*Quantization Default*), que contiene los parámetros de cuantificación. Estos dos marcadores pueden aparecer en cualquier posición de la cabecera principal, no importa en qué orden pero deben aparecer.

El resto del *code-stream*, hasta la aparición del marcador EOC (*End Of Code-stream*), que debe aparecer justo al final, está organizado como se muestra en la figura 5. Para cada *tile* de la imagen, hay un conjunto de datos. Este conjunto de datos se divide en uno o más *tile-parts*. Cada *tile-part*

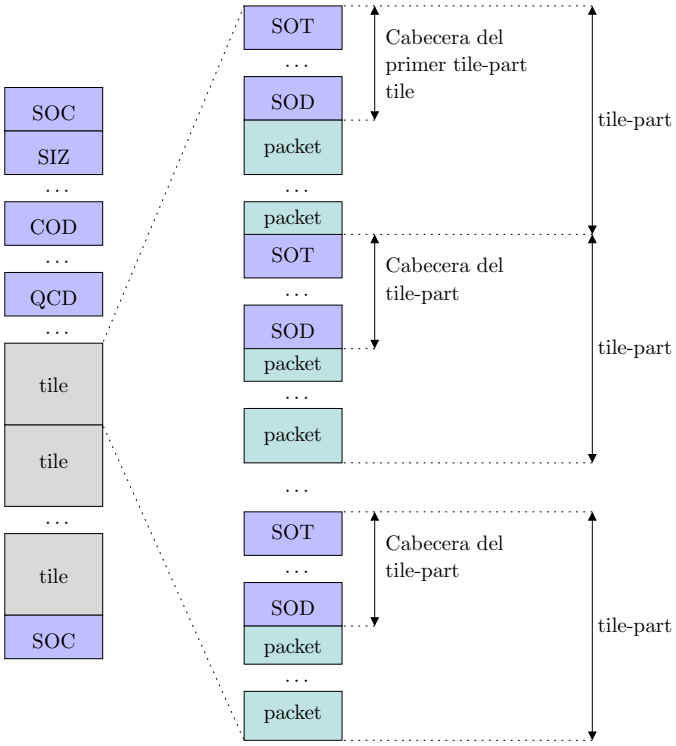


Figura 5. Organización del code-stream de un fichero JPEG 2000.

está compuesto por una cabecera y un conjunto de paquetes. La cabecera del primer *tile-part* es la cabecera principal del *tile*. La cabecera de cada *tile-part* empieza con el marcador SOT (*Start Of Tile*) y termina con el marcador SOD (*Start Of Data*), que es cuando empieza la secuencia de paquetes. La cabecera principal termina cuando se encuentra el primer marcador SOT.

Con el fin de permitir un acceso aleatorio a los datos del *code-stream*, JPEG 2000 ofrece la posibilidad de incluir los marcadores TLM (*Tile-part Lengths, Main header*), PLM (*Packet Length, Main header*) y/o PLT (*Packet Length, Tile-part header*). Los marcadores TLM y PLM se incluyen en la cabecera principal, mientras que el marcador PLT se incluye en la cabecera de un *tile* o un *tile-part*. El objetivo de los marcadores TLM es almacenar el tamaño de cada *tile-part* que aparece en el *code-stream*. Este tamaño incluye la cabecera y el conjunto de paquetes, de modo que para conocer dónde está el inicio de los datos es necesario analizar primeramente la cabecera. El marcador PLM almacena el tamaño de cada paquete de cada *tile-part* de cada *code-stream*. Cada paquete del *code-stream* tiene un determinado tamaño, que no puede ser conocido a priori. Por lo tanto la inclusión

Cuadro I
MARCADORES UTILIZADOS EN EL *code-stream* DE JPEG 2000.

	Nombre	**Valor**
Delimiting Marker Segments		
Start of Codestream	SOC	0xFF4F
Start of tile-part	SOT	0xFF90
Start of data	SOD	0xFF93
End of code-stream	EOC	0xFFD9
Fixed Info Marker Segments		
Image and tile size	SIZ	0xFF51
Functional Marker Segments		
Coding style default	COD	0xFF52
Coding style component	COC	0xFF53
Region of interest	RGN	0xFF5E
Quantization default	QCD	0xFF5C
Quantization component	QCC	0xFF5D
Progression order default	POD	0xFF5F
Pointer Marker Segments		
Tile-part lengths, main header	TLM	0xFF55
Packet length, main header	PLM	0xFF57
Packet length, tile-part header	PLT	0xFF58
Packed packet header, main header	PPM	0xFF60
Packed packet header, tile-part header	PPT	0xFF61
In Bitstream Marker Segments		
Start of packet	SOP	0xFF91
End of packet header	EPH	0xFF92
Informational Marker Segment		
Comment and extension	CME	0xFF64

de este marcador facilita el acceso aleatorio a los paquetes. El marcador PLT tiene la misma función que el marcador PLM, pero a nivel de *tile-part*, de modo que éste almacena el tamaño de todos los paquetes que pertenecen a un *tile-part*. Este marcador es más usado que el marcador PLM.

Los marcadores PLM y PLT producen un incremento del tamaño del *code-stream*, además la forma de codificar el tamaño de los paquetes ayuda a evitar excesivo *overhead*: un tamaño L de un determinado paquete, que puede ser representado con B_L *bits*, y es almacenado/codificado con $\left\lceil \frac{B_L}{7} \right\rceil$ *bytes*. Para un tamaño L se genera una secuencia de *bytes* donde solamente son usados los 7 *bits* menos significativos. El *bit* más significativo de cada *byte* indica si es (1) o no (0) el último de la secuencia. Esta forma de codificar es bastante utilizada en la Parte 9 del estándar, especialmente con el protocolo JPIP, donde se le conoce como VBAS (*Variable Byte-Aligned Segment*).

III-D. Progresiones

Los paquetes generados en el proceso de compresión JPEG 2000 no son independientes ni autosuficientes. Con un solo paquete y sin ningún tipo información adicional, es imposible averiguar a que parte de la imagen pertenece dicho paquete. El tamaño del paquete no puede ser determinado antes de realizar el proceso de compresión, y muchos paquetes no pueden ser descomprimidos si previamente no se han descomprimido otros paquetes primero. Este es el principal motivo por el que es necesario el uso de marcadores como TLM, PLT o PLM, que nos permiten acceder de forma aleatoria a los datos sin tener que descomprimirlos.

Los paquetes de cada *tile-part* aparecen de acuerdo a la progresión especificada por el último marcador COD o POD que se ha leído, antes del marcador SOD. La Parte 1 del estándar JPEG 2000 define cinco posibles tipos de progresiones para ordenar los paquetes dentro de un *tile* o *tile-part*. Cada progresión se identifica por medio de una combinación de cuatro letras: “L” para las capas de calidad, “R” para los niveles de resolución, “C” para las componentes y “P” para los *precincts*. Cada letra identifica la partición de la progresión. A continuación se muestran las diferentes progresiones permitidas en el estándar, y el orden en el que aparecerían los paquetes en cada caso.

Progresión LRCP
(*Layer-Resolution-Component-Position*)

for each layer l
  for each resolution r
    for each component c
      for each precinct p
        include the packet $\zeta_{t,c,r,p,l}$

Progresión RLCP
(*Resolution-Layer-Component-Position*)

for each layer r
  for each resolution l
    for each component c
      for each precinct p
        include the packet $\zeta_{t,c,r,p,l}$

Progresión RPCL
(*Resolution-Position-Component-Layer*)

for each layer r
  for each resolution p
    for each component c
      for each precinct l
        include the packet $\zeta_{t,c,r,p,l}$

Progresión PCRL
(*Position-Component-Resolution-Layer*)

for each layer p
  for each resolution c
    for each component r
      for each precinct l
        include the packet $\zeta_{t,c,r,p,l}$

Progresión CPRL
(*Component-Position-Resolution-Layer*)

for each layer c
  for each resolution p
    for each component r
      for each precinct l
        include the packet $\zeta_{t,c,r,p,l}$

Elegir una progresión u otra depende del tipo de aplicación que se quiera desarrollar y de cómo se quiera descomprimir los paquetes. En el caso de la transmisión de imágenes, los paquetes deben seguir un orden específico o progresión cuando son transmitidos. Cuando una imagen es transmitida desde un servidor a un cliente, el principal objetivo es conseguir que el cliente pueda ver las reconstrucciones de la imagen con una calidad que vaya incrementando a media que se van recibiendo los datos. La calidad de la reconstrucción debe ser siempre la máxima posible de acuerdo a la cantidad de datos recibida.

III-E. Formatos de archivo

Aunque el *code-stream* es completamente funcional como un formato de archivo básico, éste no permite la inclusión de información adicional que podría ser necesaria en ciertas aplicaciones, por ejemplo: el uso de metadatos, información de *copyright*, o paletas de color. Haciendo uso del marcador COM podemos incluir información adicional dentro de un *code-stream*, pero sin ningún tipo de clasificación ni ordenación.

La Parte 1 del estándar define además un formato de archivo basado en "cajas" que permite incluir dentro de un mismo archivo diferentes *code-streams* y diversa información de una forma correctamente identificada. Estos archivos tienen la extensión ".JP2", extensión que además es utilizada para nombrar a este tipo de archivos.

Los archivos JP2 son fácilmente extensibles. Tienen definida una estructura básica de caja, que contiene cualquier tipo de información. Cada caja está identificada por medio de un identificador de 4 *bytes*. Un mismo archivo puede contener varias cajas con el mismo identificador. El estándar propone un conjunto inicial de cajas, que puede ser extendido de acuerdo a las necesidades de cada caso. De hecho, el formato JP2 es la base del resto de formatos y extensiones definidos en el resto de partes del estándar.

Cada caja tiene una cabecera de 8 *bytes*. Los primeros 4 *bytes*, L, forman un entero sin signo que indica el número de bytes del contenido y los siguientes 4 *bytes*, T, contienen el identificador del tipo de caja. Este identificador es tratado como una cadena ASCII de 4 caracteres. El valor de L incluye la cabecera, por lo tanto el tamaño real del contenido de la caja es $L - 8$. L puede tener cualquier valor mayor o igual a 8, pero también puede tener un valor igual a 1 o 0. Si $L = 1$ el tamaño del contenido de

la caja es codificado como un entero sin signo de 8 *bytes*, X, situados después de T. En este caso la cabecera ocupa 16 *bytes* y el tamaño del contenido es $X - 16$. Si $L = 0$ el tamaño del contenido de la caja no está definido, pero este caso solamente es posible para la última caja del archivo.

IV. El protocolo JPIP

La Parte 9 del estándar JPEG 2000 está casi completamente dedicada al desarrollo de sistemas de exploración de imágenes remotas. En esta parte se definen un conjunto de tecnologías (protocolos, formatos de archivo, arquitecturas, etc.) que permiten explotar de manera eficiente todas las características del sistema de compresión JPEG 2000. La principal tecnología en la que se centra la Parte 9 es el protocolo JPIP (JPEG 2000 *Interactive Protocol*).

IV-A. Arquitectura

En la figura 6 se muestra cómo es la arquitectura común para un sistema cliente/servidor que permita la exploración de imágenes remotas usando el protocolo JPIP para realizar la transmisión de las imágenes.

Como se puede observar en la figura, el cliente se divide en cuatro módulos funcionales: *Browser*, Cliente JPIP, Descompresor y Caché. El módulo "*Browser*" es la interfaz que interacciona con el usuario y donde éste puede especificar una WOI de la imagen. El protocolo JPIP permite el uso de múltiples parámetros para definir la WOI que se debe transmitir, estos parámetros son: número de capas de calidad, número de componentes, etc. El protocolo JPIP impone que las WOIs tienen que estar definidas sobre regiones rectangulares dentro de un nivel de resolución determinado.

Una vez que el usuario ha definido la WOI deseada, ésta se envía a los módulos "Cliente JPIP" y "Descompresor". El primero de ellos es el encargado de la comunicación con el servidor JPIP y para ello hace uso de los mensajes y sintaxis definida en el protocolo. Cuando el servidor recibe esta información, extrae los datos necesarios de la imagen para poder hacer la reconstrucción de la WOI solicitada, y enviarlos al cliente. Estos datos se envían encapsulados en *data-bins*. En el protocolo JPIP los diferentes elementos de las particiones de una imagen JPEG 2000 son reordenados y encapsulados en *data-bins*, siendo éstos la mínima unidad de transmisión.

A medida que el cliente recibe los *data-bins* del servidor, los almacena en una caché interna (módulo "Caché"). La caché del cliente también está organizada en *data-bins*. Esta caché se utiliza continuamente por del módulo "Descompresor" para generar reconstrucciones progresivas de la WOI, que son enviadas al módulo "*Browser*" para que pueda ser visualizada por el usuario.

Excepto el módulo "Caché" que actúa como un contenedor, el resto de módulos del cliente trabajan de forma paralela. El "*Browser*" está continuamente informando al "Cliente JPIP" la WOI que ha solicitado el usuario. El "Cliente JPIP" mantiene la comunicación con el servidor para enviar la WOI y cuando recibe los datos los guarda en la caché. Los módulos comienzan su ejecución cuando el usuario define la primera WOI, y se detienen cuando se han recibido todos los datos de la WOI actual. El usuario puede indicar una nueva WOI siempre que quiera, sin tener que esperar a recibir todos los datos de la anterior petición.

La memoria caché del cliente almacena todos los *data-bins* recibidos desde el servidor, para todas las WOIs de todas las imágenes que han sido exploradas por el usuario. El servidor mantiene un modelo con el contenido de esta caché, esto le permite no tener que volver a enviar al cliente datos que ya han sido enviados anteriormente. Por ejemplo, supongamos que un usuario después de la exploración de la WOI_A, solicita una nueva WOI_B que se solapa con la WOI_A. Si el servidor mantiene actualizado el modelo de caché del cliente, cuando se solicita la WOI_B, el servidor solamente envía los *data-bins* necesarios para reconstruir la parte de WOI_B que no es común con WOI_A. Si todos los *data-bins* que permiten reconstruir una cierta WOI_i se definen como $D(WOI_i)$, podemos decir que el servidor envía los *data-bins* $D(WOI_B) - (D(WOI_A) \bigcap D(WOI_B))$.

IV-B. Partición en Data-bins

Los *data-bins* encapsulan diferentes elementos de la partición de una imagen JPEG 2000. Esta nueva partición de datos definida en el protocolo JPIP permite identificar y recopilar las diferentes partes de una imagen para su transmisión.

Cuando se inicia una comunicación cliente/servidor, se define qué tipo de *stream* de *data-bins* se va a utilizar. El protocolo JPIP

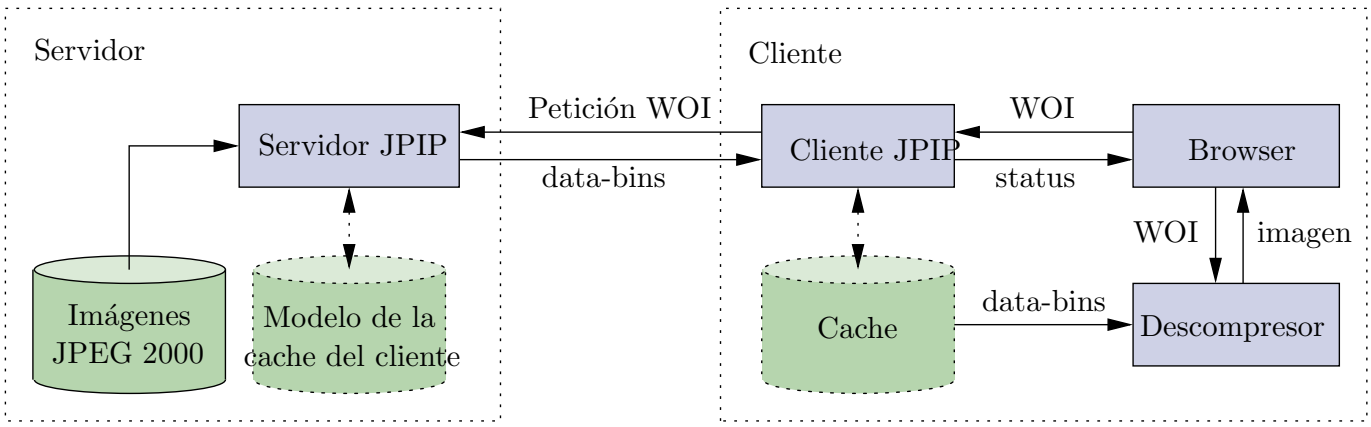


Figura 6. Arquitectura cliente/servidor del protocolo JPIP.

define dos tipos de *streams*, JPP, orientado a *precincts*, y JPT, orientado a *tiles*. El tipo de *stream* indica qué tipo de *data-bins* se deben usar para la transmisión de datos. El más utilizado es el tipo JPP.

Cada *data-bin* está identificado de manera inequívoca por la imagen o *code-stream* al que pertenece, el tipo de *data-bin* y su identificador. Esta información está incluida en la cabecera de cada *data-bin*.

Los *data-bins* pueden ser segmentados para su transmisión. Un determinado conjunto de *data-bins* se puede dividir en un número aleatorio de segmentos, y éstos se pueden enviar en cualquier orden. Cada segmento de *data-bin* incluye la información necesaria para su identificación.

En la tabla II se puede observar la relación entre los diferentes tipos de *data-bins* definidos en el protocolo JPIP. A continuación se explican los principales tipos de *data-bins*:

- *Precinct data-bin:* Contiene todos los paquetes para un nivel de resolución r, una componente c y un *tile* t. Los paquetes están ordenados dentro del *data-bin* por capas de calidad, en orden creciente.
  Cada *precinct data-bin* es identificado por un índice I que se obtiene por medio de la siguiente expresión:

$$I = t + ((c + (s \times N_c)) \times N_t)\,.$$

  N_c y N_t son el número de componentes y el número de *tiles* de la imagen a los que pertenece el *precinct*. El índice del *tile*, t al igual que e índice de la componente c y el valor s se inicializan con el valor cero. A cada *precinct* de cada *tile-component*, de todos los niveles de resolución, se le asigna un único número s de la secuencia. A todos los *precincts* del nivel de resolución más bajo se le asigna un número de secuencia que empieza en cero, situado en la esquina superior izquierda, y se continúa incrementando por columna y fila respectivamente.
- *Tile data-bin:* Contiene todos los paquetes y marcadores asociados a un *tile*. Está compuesto por la concatenación de todos los *tile-parts* asociados a un *tile*, incluyendo los marcadores SOT, SOD y todo el resto de marcadores relevantes. Los *data-bins* de este tipo se identifican por medio de un índice incremental, que se inicializa en cero para el *tile* situado en la esquina superior izquierda.
- *Tile header data-bin:* Contiene todos los marcadores asociados a un determinado *tile*. Está formado por todos los marcadores de todas las cabeceras de todos los *tile-parts* de un *tile*, sin incluir los marcadores SOT ni SOD. El identificador de este *data-bin* tiene la misma numeración que la descrita en el caso anterior.
- *Header data-bin:* Contiene la cabecera principal del *code-stream*, desde el marcador SOC (incluido) hasta el primer marcador SOT (no incluido). Los marcadores SOD y EOC no están incluidos.
- *Meta data-bins:* Estos *data-bins* aparecen solamente si la imagen asociada es un archivo de

Cuadro II
LISTA DE *data-bins* DEFINIDOS EN EL PROTOCOLO JPIP.

Data-bin	***Stream***	**Información que contiene**
Precinct	JPP	Todos los paquetes de un *precinct*.
Precinct extended	JPP	El mismo contenido con contenido adicional.
Tile	JPT	Todos los paquetes y marcadores de un *tile*.
Tile extended	JPT	El mismo contenido con contenido adicional.
Tile header	JPT	Los marcadores de la cabecera de un *tile*.
Header	JPP/JPT	Los marcadores de la cabecera de un *code-stream*.
Meta-data	JPP/JPT	Metadatos de una imagen.

la familia JP2. Los *meta data-bins* contienen un conjunto de cajas del archivo de la imagen. El estándar no define como deben ser los identificadores de estos *data-bins* ni dentro de qué cajas deben ser almacenados, pero cuando un *meta data-bin* tiene el identificador cero, quiere decir que este incluye todas las cajas contenidas en la imagen.

IV-C. Sesiones y canales

Una petición de un cliente al servidor puede ser con estado o sin estado. Las peticiones con estado se llevan a cabo dentro del contexto de una sesión de comunicación, en la que el estado de la comunicación es mantenido por el servidor. Las peticiones sin estado no requieren el uso de ninguna sesión. El uso de sesiones mejora el rendimiento del servidor porque, por ejemplo, cuando establecemos una sesión con un determinado archivo de una imagen, el servidor abre el archivo y lo prepara para poder enviarlo en *data-bins*, así que todas las peticiones asociadas a la misma sesión van a permitir al servidor tener que evitar volver a repetir el mismo proceso para cada una de ellas. Las peticiones sin estado se pueden considerar como sesiones únicas que terminan cuando el servidor envía la respuesta al cliente.

Bajo una sesión, usando peticiones con estado, el cliente puede abrir múltiples canales, siendo capaz de realizar múltiples peticiones con estado asociadas a la misma sesión. Este comportamiento es especialmente útil para aplicaciones que muestran de forma simultánea diferentes regiones de interés de la misma imagen. Los canales se pueden abrir y cerrar de forma independiente, sin que afecte a la sesión. Para cerrar una sesión habría que cerrar todos los canales abiertos asociados a la misma.

Cada sesión tiene asociado un conjunto de imágenes. Una sesión implica que el servidor tiene que mantener un modelo de la caché del cliente. Este modelo solamente se mantiene mientras que la sesión permanezca activa. Un canal, para una determinada sesión, está asociado a una determinada imagen y a un tipo específico de *data-bin stream* (JPP o JPT). El canal se identifica de manera inequívoca por medio de un código alfanumérico asignado por el servidor. Las sesiones no se identifican, ya que el identificador de canal debe ser suficiente para identificar el canal, así como la sesión a la que pertenece.

Muchas veces los clientes están interesados en mantener en el servidor su modelo de caché entre diferentes sesiones. El servidor por defecto no permite esta posibilidad, pero el protocolo JPIP define una serie de mensajes que permiten al cliente modificar el modelo que el servidor tiene de su caché. Por lo tanto, cuando un cliente inicia una nueva sesión con el servidor, podría enviar un resumen del contenido actual de su caché para mejorar la comunicación con el servidor y evitar el reenvío de datos de los que ya dispone en su caché.

Los clientes pueden indicar en sus peticiones un parámetro que le indique al servidor cuál debe ser el tamaño máximo de la respuesta. Gracias al modelo de caché existente en el servidor, el cliente puede realizar la misma petición de forma sucesiva pero variando los parámetros, de modo que el servidor le puede enviar una respuesta de forma incremental. Por lo tanto, el cliente tiene cierto control y puede adaptar el intercambio de información dependiendo del ancho de banda disponible y de la latencia de la red, pero hay que tener en cuenta que el servidor puede modificar estos parámetros y no devolver exactamente lo que el cliente había solicitado. Este modo de comunicación es el más común en las implementaciones existentes de JPIP.

IV-D. Mensajes

Las peticiones JPIP están formadas por una secuencia ASCII de pares "parámetro = valor". Esto permite que una petición JPIP pueda ser encapsulada dentro de un mensaje GET del protocolo HTTP, junto al carácter '?', concatenando todos los pares con el símbolo '&'.

Algunos de los parámetros disponibles para enviar una petición JPIP son los siguientes:

- **"fsiz=R_x,R_y":** Se utiliza para identificar la resolución correspondiente a la ventana seleccionada solicitada. El servidor elige el mayor nivel de resolución de la imagen tal que sus dimensiones $R'_x \times R'_y$ satisfagan que $R'_x \geq R_x$ y $R'_y \geq R_y$. Por lo general este parámetro incluye la resolución de pantalla del usuario.
- **"roff=P_x,P_y":** Se utiliza para identificar la esquina superior izquierda (desplazamiento) de la región espacial correspondiente a la ventana seleccionada solicitada. Si no se usa este campo, el servidor asume que el valor por defecto del desplazamiento es 0.
- **"rsiz=S_x,S_y":** Se utiliza para identificar el tamaño horizontal y vertical de la región espacial correspondiente a la ventana seleccionada solicitada. De no haberlo, la región va hasta la esquina inferior derecha de la imagen.
- **"len=número de *bytes*":** El cliente informa con este parámetro al servidor el número máximo de bytes que pueden ser incluidos en la respuesta. El servidor tiene en cuenta este límite para la respuesta actual y para todas las respuestas que se realicen dentro de la misma sesión.
- **"target=imagen":** Este parámetro identifica el archivo de la imagen de la que queremos obtener los datos. Cuando el protocolo usado es HTTP, no es obligatorio especificarlo porque el nombre del archivo de la imagen se puede obtener de la propia URL utilizada en el mensaje GET.
- **"cnew=protocolo":** Se utiliza cuando el cliente quiere abrir un nuevo canal dentro de la misma sesión. Hay que indicar el protocolo que se va a utilizar en este nuevo canal. Los distintos protocolos que podemos usar son "http" y "http-tcp".
- **"cid=identificador de canal":** Cuando un cliente crea un nuevo canal, el servidor envía el identificador de canal y éste debe ser utilizado en todas las peticiones asociadas a este canal.
- **"cclose=identificador de canal":** Este parámetro se utiliza cuando un cliente quiere cerrar un determinado canal. Solamente hay que indicar el identificador del canal que se quiere cerrar.
- **"type=tipo de *stream*":** Cuando se crea un canal, el cliente debe indicar el tipo de *data-bin stream* que se va a utilizar. Los principales tipos que se pueden utilizar son JPP ("jpp-stream") y JPT ("jpt-stream").
- **"model=...":** Como hemos comentado anteriormente, el cliente puede necesitar informar al servidor del contenido de su caché para que actualice su modelo de caché. Por ejemplo, `model=Hm,H*,M2,P0:20` quiere decir al servidor que incluya en el modelo de caché la cabecera principal del *code-stream*, las cabeceras de todos los *tiles*, el *meta data-bin* número 2 y los primeros 20 *bytes* del *precinct* 0.

V. Trabajos previos

La actualización condicional, a la que llamaremos partir de ahora en este documento sólo por CR (*Conditional Replenishment*), ha sido utilizada con éxito desde hace algunos años para codificar secuencias de imágenes con poco movimiento.

La primera referencia data de finales de los sesenta [7], donde Mounts propuso un método para la codificación de señales de televisión en la que se hacía uso de la correlación temporal existente entre los *frames* consecutivos, con el objetivo de reducir el *bit-rate* utilizado en la transmisión de la señal. La técnica propuesta solamente codificaba aquellos elementos que habían cambiado entre *frames* consecutivos en lugar de codificar todos los elementos que componían cada *frame*.

En [8] a finales de los noventa, McCanne *et al.* exploraron la posibilidad de utilizar un codificador de vídeo basado en CR para el MBone (*Multicast Backbone*), gracias a su reducida complejidad y su resistencia a la pérdida de datos.

Más recientemente, fueron estudiadas otras posibles soluciones que ofrecían buenos resultados desde el punto de vista de la compresión y decodificación. En [9], Cheung y Ortega investigaron el rendimiento de un codec de vídeo predictivo con compensación de movimiento que estaba basado en la codificación de vídeo distribuido y permitía la reproducción del vídeo hacia delante y hacia detrás.

En 2009 [10], Devaux *et al.* propusieron una solución basada en CR para explotar la redundancia temporal de secuencias de vídeo-vigilancia que previamente habían sido codificadas en formato JPEG 2000. Además esta aproximación no se basaba en la compensación de movimiento. En este caso el emisor (servidor) tiene que determinar, y enviar al receptor (cliente), los puntos óptimos a los cuales es mejor usar los datos del *code-stream* JPEG 2000 que la predicción (generada con la reconstrucción del último *frame* y una estimación del *background*).

Finalmente, también desde 2009 el trabajo de Naman *et al.* es similar en esencia, pero en este caso, se propone el uso de CR con referencias a múltiples *frames* y el uso de compensación de movimiento para codificar aquellas secuencias de imágenes con una gran cantidad de movimiento [11] y la misma arquitectura pero sin compensación de movimiento para secuencias de imágenes más estáticas [12].

Después de un análisis de todas las aproximaciones que hemos comentado anteriormente, se puede afirmar que nuestro trabajo está motivado por las siguientes razones:

1. Todas las aproximaciones propuestas necesitan algún procesamiento previo sobre la secuencia de imágenes en el lado del servidor y por lo tanto la secuencia debe ser conocida a priori, y en algunos casos incluso el orden de las imágenes. En el caso de la aplicación *JHelioViewer* esto es imposible de realizar porque sólo durante la interacción con el usuario se define la composición de las secuencias de imágenes.
2. Los recursos computacionales requeridos por el emisor para realizar una transmisión en tiempo real son extremadamente altos debido a la gran cantidad de datos que debe ser manejada (recordemos que las imágenes con las que estamos trabajando son imágenes de 4096×4096 *píxeles*, y necesitamos un *frame-rate* de al menos 20 imágenes por segundo).

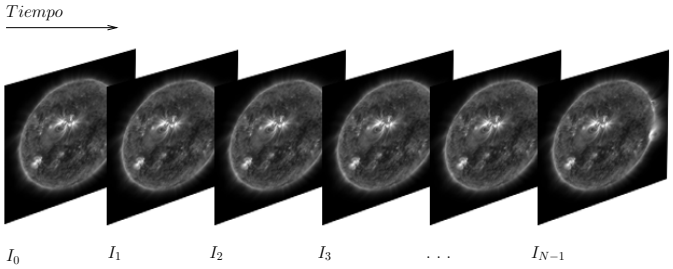


Figura 7. Una secuencia de imágenes capturadas en diferentes instantes de tiempo. El subíndice denota el instante de muestreo (uniforme) de la imagen en cuestión.

3. Ninguna de las aproximaciones anteriores es completamente compatible con el estándar JPIP, porque requieren que ciertos datos adicionales sean transmitidos desde el servidor hasta los clientes, y el protocolo JPIP no permite esto (excepto en forma de imágenes comprimidas).

Por lo tanto, ninguna de las soluciones existentes puede ser fácilmente utilizada para realizar un sistema de exploración interactiva de imágenes remotas como *JHelioviewer*. Además, en [13] se propone una estrategia de *prefetching* que podría mejorar la experiencia del usuario y esta solución es totalmente compatible con la propuesta que hacemos en este trabajo, pero podría ser bastante costoso y de difícil implementación con las aproximaciones que se han descrito anteriormente.

VI. Propuesta

Como ya se ha explicado previamente, en las secuencias de imágenes donde existen regiones que no cambian durante un largo período de tiempo, podemos hacer uso de CR. En nuestro caso estamos trabajando con imágenes solares que han sido capturadas con muy pocos segundos de diferencia entre ellas, lo que nos va a permitir que el uso de CR aproveche toda la redundancia temporal que pueda existir entre ellas.

Por otra parte, gracias a la combinación de CR con las características que nos ofrece el estándar JPEG 2000, hemos conseguido diseñar un algoritmo que permite hacer la reconstrucción en tiempo real de una secuencia de imágenes con una resolución de 4096×4096 píxeles usando un *frame-rate* de 25 imágenes por segundo en una máquina con

modestos recursos computacionales y un limitado ancho de banda.

En la descripción de nuestra propuesta y con el objeto de simplificar la exposición, vamos a suponer que el ancho de banda del canal de comunicación entre el servidor JPIP y el cliente, al igual que el *frame-rate* seleccionado por el usuario, permanecen constantes durante todo el proceso de visualización de la secuencia de vídeo. Debe de tenerse en cuenta, sin embargo, que este hecho no afectaría a la descripción del algoritmo ni a la eficiencia del mismo.

Antes de describir el algoritmo propuesto vamos a repasar la notación que se ha utilizado. La secuencia de imágenes ha sido denotada como I y I_n será la n-ésima imagen de la secuencia, donde $0 \leq n < N$. Las imágenes que finalmente son visualizadas con la técnica de actualización condicional han sido denotadas como I'.[2]

Como ya se ha indicado en la Sección III, debido a una de las características que nos ofrece el estándar JPEG 2000, las imágenes pueden ser descomprimidas desde el mismo *code-stream* con diferentes niveles de resolución espacial. Esta característica es uno de los puntos claves de nuestra propuesta. En nuestro caso, vamos a definir $I_n = I_n^0$ como la imagen a la máxima resolución espacial (véase la Figura 8) y I_n^r será una versión escalada a otro nivel de resolución de la imagen I_n, que tiene una resolución de $S/2^r$ y $0 \leq r < R+1$, donde S es el tamaño original de la imagen y R es el número de etapas utilizado a la hora de aplicar la transformada DWT, durante el proceso de compresión de la imagen.

[2]Debe tenerse en cuenta que al tratarse de una visualización progresiva en el tiempo, sólo cuando la transmisión de cada imagen ha terminado se verifica que las imágenes reconstruidas son idénticas a las originales. Mientras dicha transmisión no ocurra, el usuario remoto lo que ve siempre es sólo una aproximación (en calidad incremental en el tiempo) de las imágenes.

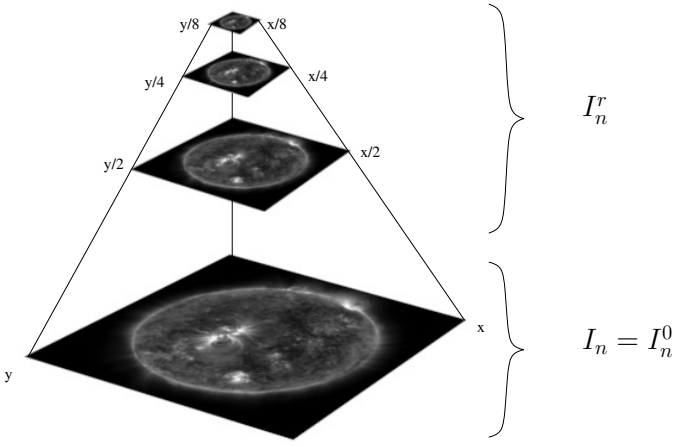


Figura 8. Diferentes niveles de resolución espacial sobre un mismo flujo de datos JPEG 2000. Para este ejemplo se ha supuesto que la DWT se ha aplicado, al menos, 3 veces.

Cuadro III
EJEMPLO DE LOS NIVELES DE RESOLUCIÓN ESPACIAL PARA UNA IMAGEN CON UN TAMAÑO ORIGINAL DE 4096×4096 PÍXELES ($S = 4096$) Y COMPRIMIDA CON 7 ETAPAS DWT ($R = 7$).

r	I_n^r	$S/2^r$	
0	I_n^0	$4096/2^0 =$	4096
1	I_n^1	$4096/2^1 =$	2048
2	I_n^2	$4096/2^2 =$	1024
3	I_n^3	$4096/2^3 =$	512
4	I_n^4	$4096/2^4 =$	256
5	I_n^5	$4096/2^5 =$	128
6	I_n^6	$4096/2^6 =$	64
7	I_n^7	$4096/2^7 =$	32

Para simplificar la descripción del algoritmo asumimos que las imágenes con las que vamos a trabajar son cuadradas, por lo tanto sólo será necesario utilizar un único parámetro ($S/2^r$) para determinar el número de píxeles utilizado en cada nivel de resolución espacial.

En la Tabla III, a modo de ejemplo, se pueden ver los distintos niveles de resolución que podemos obtener para una imagen con un tamaño original de 4096×4096 píxeles ($S = 4096$) y comprimida con 7 etapas *wavelet* ($R = 7$). Como se puede apreciar, tenemos que $0 \leq r < 8$, y por lo tanto, la imagen de mayor resolución espacial será I_n^0 de 4096×4096 píxeles y la de menor resolución espacial I_n^7 de 32×32 píxeles.

El último parámetro que aparece en la descripción del algoritmo es el parámetro Q, que hace referencia al número de capas de calidad con el que han sido comprimidas cada una de las imágenes de la secuencia.

VI-A. Algoritmo

Todas las imágenes de la secuencia de vídeo se van a transmitir desde el servidor hasta los clientes siguiendo los criterios de una progresión LRCP, lo que quiere decir que vamos a ir recibiendo poco a poco las diferentes capas de calidad con las que se ha comprimido la imagen, y progresivamente iremos mejorando la calidad de la misma. Cuanto mayor sea el número de capas de calidad recibidas mucho

mayor será la calidad de la imagen visualizada al comienzo de la transmisión, aunque los *code-streams* crecerían ligeramente.

Nuestro algoritmo trabaja de la siguiente manera: el servidor envía progresivamente la primera imagen de la secuencia durante un período de tiempo, y ésta es la primera imagen que se muestra en el lado del cliente. Dependiendo del ancho de banda existente entre el canal de comunicación establecido entre el servidor y el cliente, se recibirá mayor o menor cantidad de datos y por lo tanto la imagen que se visualizará por primera vez tendrá más o menos calidad.

Una vez que se han recibido los datos de la primera imagen se solicita la siguiente imagen de la secuencia, pero esta vez con el menor nivel de resolución posible (I_n^R). Gracias a una de las características del estándar JPEG 2000 que nos permite descomprimir una imagen a diferentes niveles de resolución espacial desde el mismo flujo de datos, también podemos obtener el menor nivel de resolución de la primera imagen. De esta manera podemos calcular de una forma muy sencilla cuáles son las diferencias existentes entre las dos miniaturas de las imágenes. Para conocer qué es lo que ha cambiado, vamos a calcular el Error Cuadrático Medio (MSE) entre los *precincts* de las dos imágenes, de esta manera podemos conocer de una forma rápida y computacionalmente sencilla los *precincts* que debemos solicitar al servidor de la segunda imagen y hacer uso de la técnica de actualización condicional con los datos que ya disponemos de la primera imagen de la secuencia. Los paquetes de datos JPEG 2000 que se han recibido para obtener la segunda imagen a la menor resolución no han sido enviados en vano ya que éstos son necesarios para obtener la imagen a la mayor resolución. El proceso descrito anteriormente se repite para todas las imágenes de la secuencia.

Una vez que hemos explicado cuál es la base de nuestra idea, hacemos la siguiente propuesta de esquema de transmisión de secuencias de imágenes JPEG 2000 usando actualización condicional controlada por el cliente:

1. Sea $q \leftarrow 1$ el número de capas de calidad utilizadas en las reconstrucciones actuales.
2. Sea $n \leftarrow 1$ el índice de imágenes reconstruidas.
3. Obtenemos la siguiente capa de calidad de la imagen I_{n-1} e inicializamos I'_{n-1} con este valor.
4. Obtenemos solamente la siguiente capa de calidad de la imagen n a la menor resolución espacial I_n^R (la imagen en miniatura).
5. Calculamos el Error Cuadrático Medio (MSE) entre los *precincts* de las imágenes I_{n-1}^R y I_n^R, y los almacenamos en una lista L en orden descendente (véase la Figura 9).
6. $I'_n \leftarrow I_n(L) \cup (\overline{I'_{n-1} \cap I_n(L)})$. Copiamos de la imagen I'_{n-1} a I'_n aquellos *precincts* que permanecen "constantes" y actualizamos desde la imagen I_n aquellos *precincts* que están en la lista L con una calidad q (véase la Figura 10).
7. $n \leftarrow n + 1$.
8. Si $n < N - 1$, ir al paso 4.
9. $q \leftarrow q + 1$.
10. Si $q \leq Q$, ir al paso 2.

Con el objetivo de que el lector entienda con más comodidad el funcionamiento de nuestro algoritmo, en la figura 11 se presenta un ejemplo de cómo sería la transmisión de las cinco primeras imágenes de una secuencia de imágenes cualquiera, donde solamente ocurren cambios en la región central. Las imágenes que finalmente se muestran al usuario han sido etiquetadas como "Reconstrucción con CR". Como se describe en la siguiente sección, las reconstrucciones realizadas con CR son mejores que las que se obtienen sin utilizar CR, obteniendo mejores valores de PSNR y una mejor reconstrucción visual.

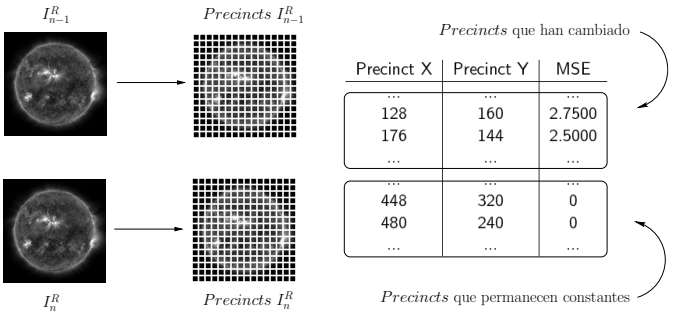


Figura 9. Etapa del algoritmo donde se realiza el cálculo del Error Cuadrático Medio (MSE) entre los *precincts* de las imágenes I_{n-1}^R y I_n^R, y se almacena el resultado en una lista L en orden descendente.

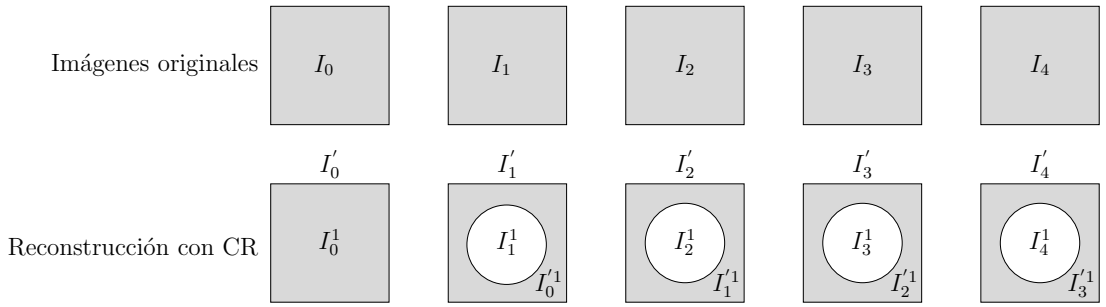


Figura 11. Ejemplo con las cuatro primeras iteraciones de nuestro algoritmo.

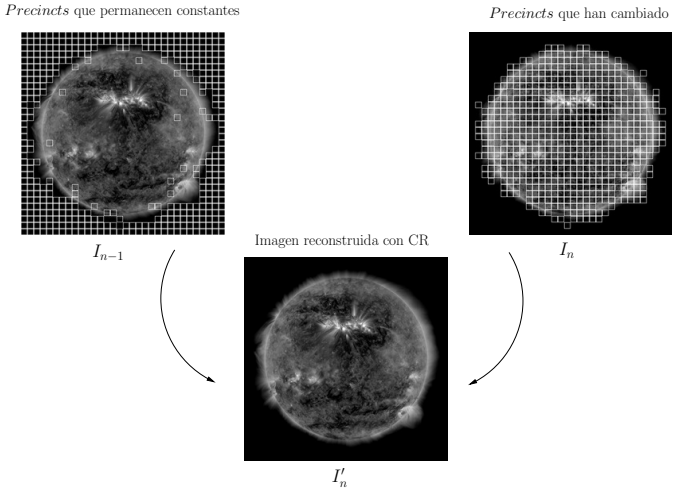


Figura 10. Etapa donde copiamos de la imagen I'_{n-1} a I'_n aquellos *precincts* que permanecen "constantes" y actualizamos desde la imagen I_n aquellos *precincts* que están en la lista L con una calidad q.

VII. Evaluación

Nuestra propuesta ha sido testeada con dos secuencias de imágenes diferentes. La primera de ellas está compuesta por un conjunto de 140 imágenes, tomadas desde el observatorio solar SDO/AIA, con una resolución espacial de 4096×4096 píxeles, y tomadas con una cadencia de 432 segundos.

La segunda secuencia se trata de *Akiyo*, una secuencia de *benchmark* conocida, formada por un conjunto de 300 imágenes con una resolución espacial de 352×288 píxeles. A diferencia de la secuencia solar, esta segunda secuencia se trata de una secuencia con muy poco movimiento global lo que la convierte en una candidata ideal para ser transmitida utilizando nuestra propuesta.

La evaluación ha consistido en la realización de múltiples simulaciones sobre las dos secuencias de imágenes que hemos mencionado anteriormente. Para cada una de la simulaciones realizadas se han ido probando diferentes configuraciones, donde hemos alterado algunos parámetros relacionados con la codificación y la transmisión, asignándole diferentes valores en cada caso. Los parámetros de codificación que hemos modificado han sido: el tamaño de *precinct*, los niveles de calidad y los niveles de resolución espacial. También hemos simulado diferentes escenarios con distinto ancho de banda para el canal de comunicación establecido entre el servidor y el cliente, con el fin de evaluar el comportamiento de nuestro algoritmo, y ver cómo reacciona ante situaciones donde disponemos de muy poco ancho de banda.

VII-A. Ejemplo 1. SDO/AIA

Los parámetros de codificación utilizados en la secuencia de este ejemplo han sido los siguientes:

- Tamaño de *precinct*; $P = 32$.
- $Q = 8$ niveles de calidad.
- $R = 7$ niveles de resolución espacial.

La transmisión de la secuencia se ha simulado con los siguientes parámetros:

- *Bit-rate:* 58×10^4 *bits* por segundo.
- *Frame-rate:* 25 *frames* por segundo

En este caso estamos simulando que tenemos muy poco ancho de banda entre cliente y servidor, lo que va a provocar que en cada iteración del algoritmo se actualicen muy pocos *precincts* de la siguiente imagen. Aún así obtenemos una mejor reconstrucción visual cuando se utiliza una transmisión controlada por el cliente con CR que cuando se utiliza una transmisión estándar (véase la Figura 12). En esta simulación, nuestra propuesta también obtiene mejores valores de PSNR, en la mayoría de reconstrucciones de la secuencia, al compararla

con las reconstrucciones obtenidas con el método tradicional sin CR (véase la Figura 13).

VII-B. Ejemplo 2. SDO/AIA

Los parámetros de codificación utilizados en la secuencia de este ejemplo han sido los siguientes:

- Tamaño de *precinct*; $P = 32$.
- $Q = 8$ niveles de calidad.
- $R = 7$ niveles de resolución espacial.

La transmisión de la secuencia se ha simulado con los siguientes parámetros:

- *Bit-rate:* 224×10^5 *bits* por segundo.
- *Frame-rate:* 25 *frames* por segundo

En este caso estamos simulando que tenemos el ancho de banda suficiente para recibir del servidor todos los *precincts* de la siguiente imagen que han cambiado con respecto a la imagen actual. La reconstrucción visual que se obtiene en esta simulación es muy similar en ambos casos. Es muy complicado apreciar alguna diferencia entre las imágenes reconstruidas a simple vista (véase la Figura 14). Los valores de PSNR que obtenemos en todas las reconstrucciones de la imagen, cuando utilizamos CR, son mucho mayores que cuando utilizamos una transmisión estándar (véase la Figura 15).

VII-C. Ejemplo 3. Akiyo

Los parámetros de codificación utilizados en la secuencia de este ejemplo han sido los siguientes:

- Tamaño de *precinct*; $P = 32$.
- $Q = 8$ niveles de calidad.
- $R = 3$ niveles de resolución espacial.

La transmisión de la secuencia se ha simulado con los siguientes parámetros:

- *Bit-rate:* 38×10^4 *bits* por segundo.
- *Frame-rate:* 25 *frames* por segundo

La secuencia utilizada en esta simulación, *Akiyo*, contiene mayor redundancia temporal que la secuencia solar, SDO/AIA, por este motivo los resultados obtenidos son mucho mejores que los obtenidos en el caso de la secuencia solar. A pesar de tener muy poco ancho de banda entre el cliente y el servidor, las reconstrucciones visuales obtenidas con la técnica de CR son notablemente mejores (véase la Figura 14). Nuestra propuesta también obtiene mejores valores de PSNR, en la gran mayoría de reconstrucciones de la secuencia, al compararla con las reconstrucciones obtenidas con el método tradicional sin CR (véase la Figura 15).

VII-D. Ejemplo 4. Akiyo

Los parámetros de codificación utilizados en la secuencia de este ejemplo han sido los siguientes:

- Tamaño de *precinct*; $P = 32$.
- $Q = 8$ niveles de calidad.
- $R = 3$ niveles de resolución espacial.

La transmisión de la secuencia se ha simulado con los siguientes parámetros:

- *Bit-rate:* 11×10^6 *bits* por segundo.
- *Frame-rate:* 25 *frames* por segundo

Igual que en el Ejemplo 2, hemos simulado que tenemos el ancho de banda suficiente para recibir del servidor todos los *precincts* de la siguiente imagen que han cambiado con respecto a la imagen actual. En este caso volvemos a obtener reconstrucciones muy parecidas, donde es muy complicado apreciar alguna diferencia a simple vista (véase la Figura 14). Desde el punto de vista del PSNR obtenemos valores mucho más altos cuando hacemos uso de CR (véase la Figura 15).

VII-E. Ejemplo 5

En este ejemplo queremos mostrar los valores de PSNR que vamos obteniendo durante la ejecución del algoritmo, para las distintas capas de calidad (véase la Figura 20).

Los parámetros de codificación/transmisión utilizados para ambas secuencias han sido los siguientes:

- Secuencia SDO/AIA:
  - Tamaño de *precinct*; $P = 128$.
  - $Q = 8$ niveles de calidad.
  - $R = 8$ niveles de resolución espacial.
  - *Bit-rate:* 27×10^6 *bits* por segundo.
  - *Frame-rate:* 25 *frames* por segundo
- Secuencia *Akiyo*:
  - Tamaño de *precinct*: $P = 32$.
  - $Q = 8$ niveles de calidad.

- $R = 3$ niveles de resolución espacial.
- *Bit-rate:* 8×10^5 *bits* por segundo.
- *Frame-rate:* 25 *frames* por segundo

VIII. Conclusiones y trabajo futuro

Este trabajo muestra cómo se puede mejorar la visualización de secuencias de imágenes JPEG 2000 haciendo uso de la técnica de actualización condicional controlada exclusivamente por el cliente. El sistema propuesto tiene la gran ventaja de que es completamente compatible con el estándar JPIP porque lo único que hacemos con nuestro algoritmo es cambiar el orden de los datos que queremos obtener desde el servidor, de modo que siempre recibimos primero los paquetes de aquellas regiones de la imagen donde se hayan producido más cambios.

Como trabajo futuro, y con la idea de mejorar la calidad de las reconstrucciones de aquellas secuencias con un alto grado de movimiento, se podrían realizar predicciones de compensación de movimiento en el lado del cliente. Esta mejora es totalmente compatible con nuestra propuesta y no afectaría en ningún momento la compatibilidad del algoritmo con el estándar JPEG 2000 y su protocolo de transmisión asociado JPIP.

Agradecimientos

Me gustaría agradecer la inestimable ayuda que me han ofrecido Juan Pablo García Ortiz y Vicente González Ruiz, con la cual el desarrollo de este trabajo ha sido mucho más rápido y sencillo. Gracias.

Bibliografía

[1] International Organization for Standardization, "Information Technology - JPEG 2000 Image Coding System - Core Coding System," ISO/IEC 15444-1:2004, September 2004.

[2] ——, "Information Technology - JPEG 2000 Image Coding System - Interactivity Tools, APIs and Protocols," ISO/IEC 15444-9:2005, November 2005.

[3] D. Müller, B. Fleck, G. Dimitoglou, B. W. Caplins, D. E. Amadigwe, J. P. G. Ortiz, A. A. B. Wamsler, V. K. Hughitt, and J. Ireland, "JHelioviewer: Visualizing large sets of solar images using JPEG 2000," *Computing in Science and Engineering*, vol. 11, no. 5, pp. 38–47, September 2009.

[4] "European Space Agency," http://www.esa.int.

[5] "National Aeronautics and Space Administration," http://www.nasa.gov.

[6] W. Pesnell, "The Solar Dynamics Observatory: Your eye on the Sun," in *37th COSPAR Scientific Assembly*, ser. COSPAR, Plenary Meeting, vol. 37, 2008, pp. 2412–+.

[7] F. Mounts, "A video encoding system with conditional picture-element replenishment," *Bell System Technical Journal*, vol. 48, pp. 2545 – 2554, September 1969.

[8] S. McCanne, M. Vetterli, and V. Jacobson, "Low-complexity video coding for receiver-driven layered multicast," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 6, pp. 983 – 1001, August 1997.

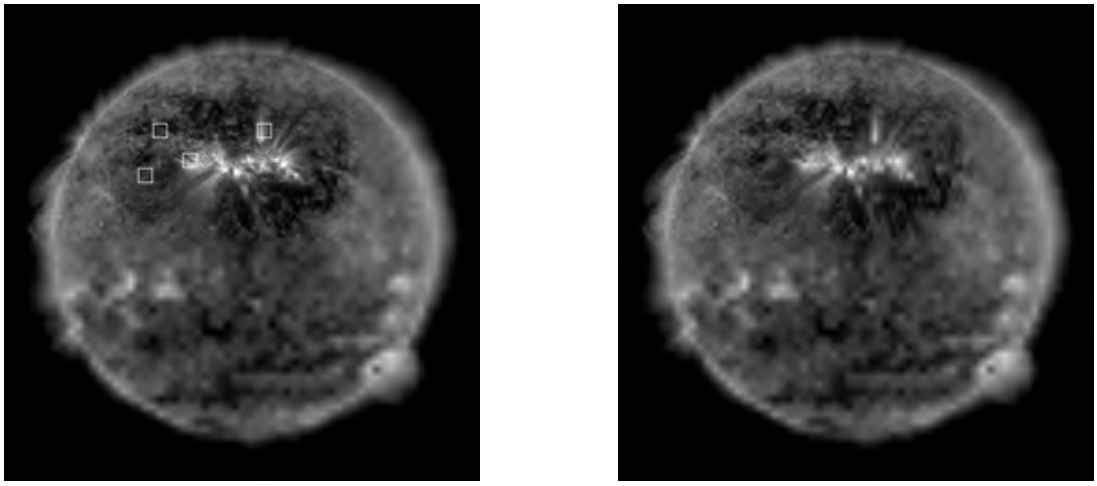
[9] N.-M. Cheung and A. Ortega, "Flexible video decoding: A distributed source coding approach," in *IEEE Workshop on Multimedia Signal Processing*, October 2007, pp. 103 – 106.

[10] F.-O. Devaux, J. Meessen, C. Parisot, J.-F. Delaigle, B. Macq, and C. D. Vleeschouwer, "Remote interactive browsing of video surveillance content based on jpeg 2000," *IEEE Transactionf on Circuits and Systems for Video Technology*, vol. 19, no. 8, pp. 1143 – 1157, August 2009.

[11] A. Naman and D. Taubman, "Rate-distortion optimized jpeg2000-based scalable interactive video (jsiv) with motion and quantization bin side-information," in *IEEE International Conference on Image Processing*, November 2009, p. 3081 – 3084.

[12] ——, "Predictor selection using quantization intervals in jpeg2000-based scalable interactive video (jsiv)," in *IEEE International Conference on Image Processing*, September 2010, pp. 2897 – 2900.

[13] J. G. Ortiz, V. G. Ruiz, I. García, D. Müller, and G. Dimitoglou, "Interactive browsing of remote jpeg 2000 image sequences," in *IEEE International Conference on Pattern Recognition*. Istambul, Turkey: IEEE press, August 2010, pp. 3179 – 3182.



con CR sin CR

Figura 12. Reconstrucción visual de una imagen de la secuencia SDO/AIA, simulando una transmisión controlada por el cliente haciendo uso de CR y una transmisión estándar, utilizando un *bit-rate* de 58×10^4 *bits* por segundo.

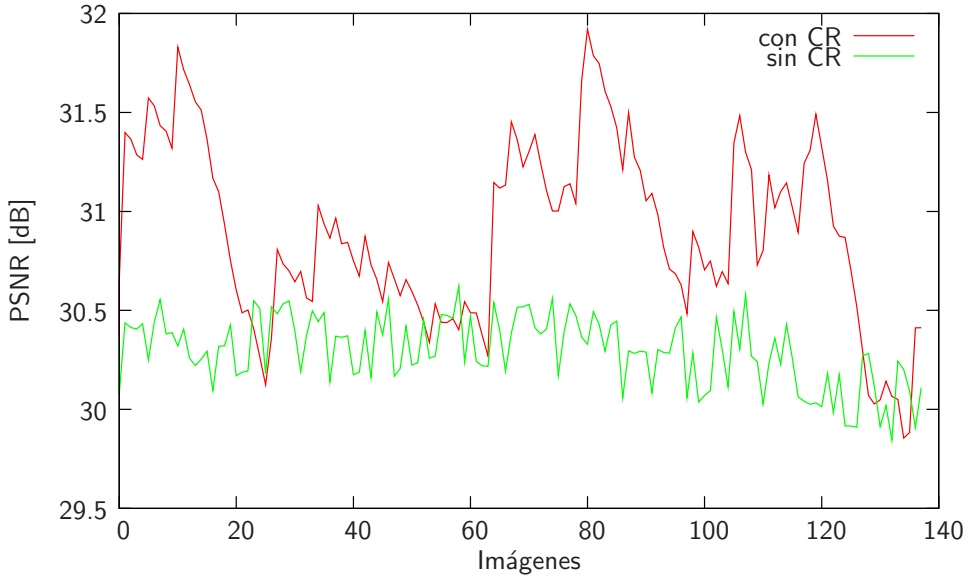
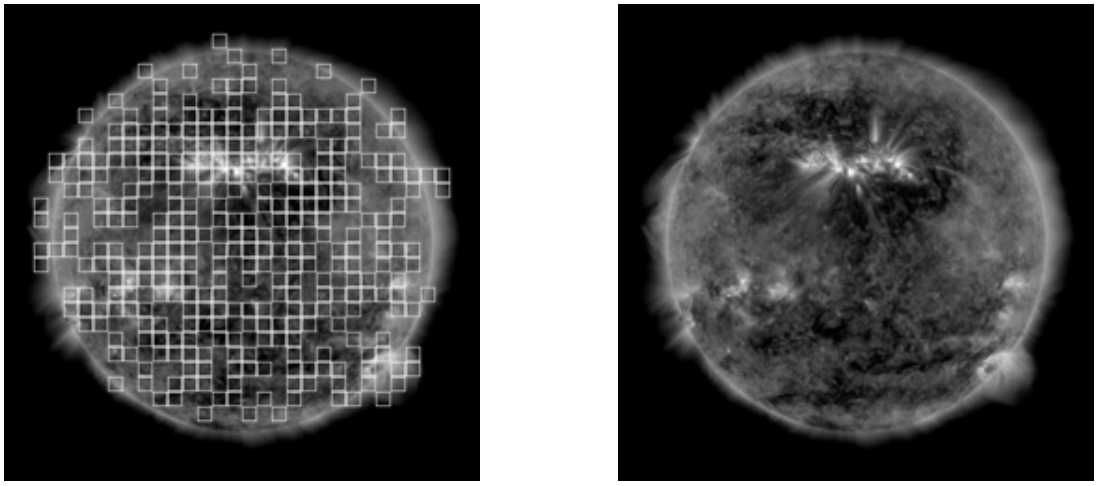


Figura 13. Gráfica que compara el PSNR obtenido para cada una de las imágenes de la secuencia SDO/AIA, cuando se utiliza una transmisión controlada por el cliente haciendo uso de CR y cuando se utiliza una transmisión estándar. El *bit-rate* empleado para la transmisión es de 58×10^4 *bits* por segundo.



con CR sin CR

Figura 14. Reconstrucción visual de una imagen de la secuencia SDO/AIA, simulando una transmisión controlada por el cliente haciendo uso de CR y una transmisión estándar, utilizando un *bit-rate* de 224×10^5 *bits* por segundo.

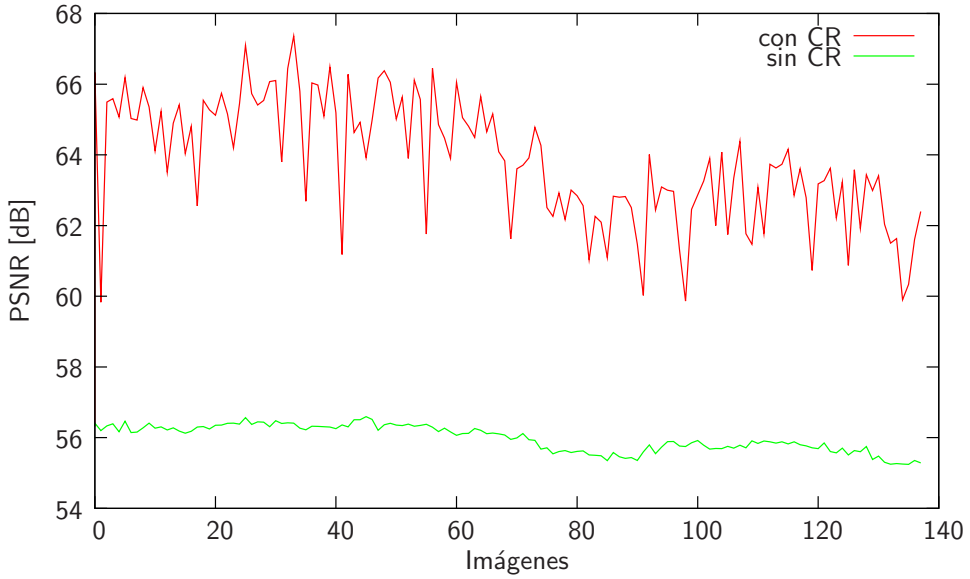


Figura 15. Gráfica que compara el PSNR obtenido para cada una de las imágenes de la secuencia SDO/AIA, cuando se utiliza una transmisión controlada por el cliente haciendo uso de CR y cuando se utiliza una transmisión estándar. El *bit-rate* empleado para la transmisión es de 224×10^5 *bits* por segundo.

con CR sin CR

Figura 16. Reconstrucción visual de una imagen de la secuencia *Akiyo*, simulando una transmisión controlada por el cliente haciendo uso de CR y una transmisión estándar, utilizando un *bit-rate* de 38×10^4 *bits* por segundo.

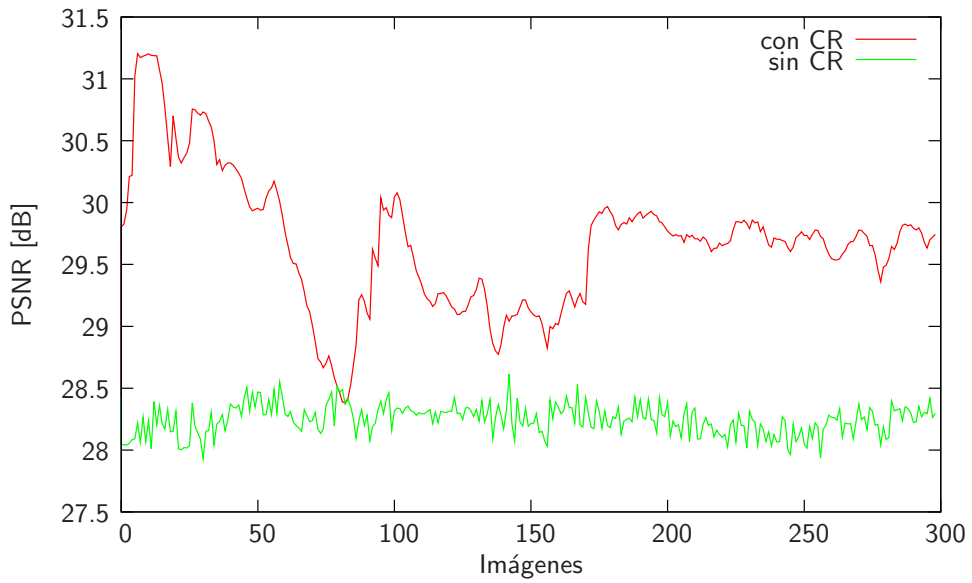
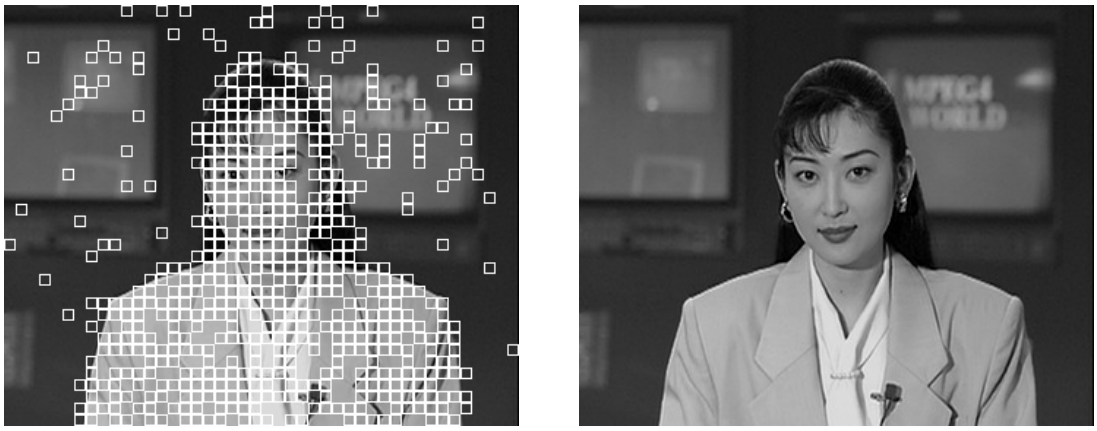


Figura 17. Gráfica que compara el PSNR obtenido para cada una de las imágenes de la secuencia *Akiyo*, cuando se utiliza una transmisión controlada por el cliente haciendo uso de CR y cuando se utiliza una transmisión estándar. El *bit-rate* empleado para la transmisión es de 38×10^4 *bits* por segundo.



con CR sin CR

Figura 18. Reconstrucción visual de una imagen de la secuencia *Akiyo*, simulando una transmisión controlada por el cliente haciendo uso de CR y una transmisión estándar, utilizando un *bit-rate* de 11×10^6 *bits* por segundo.

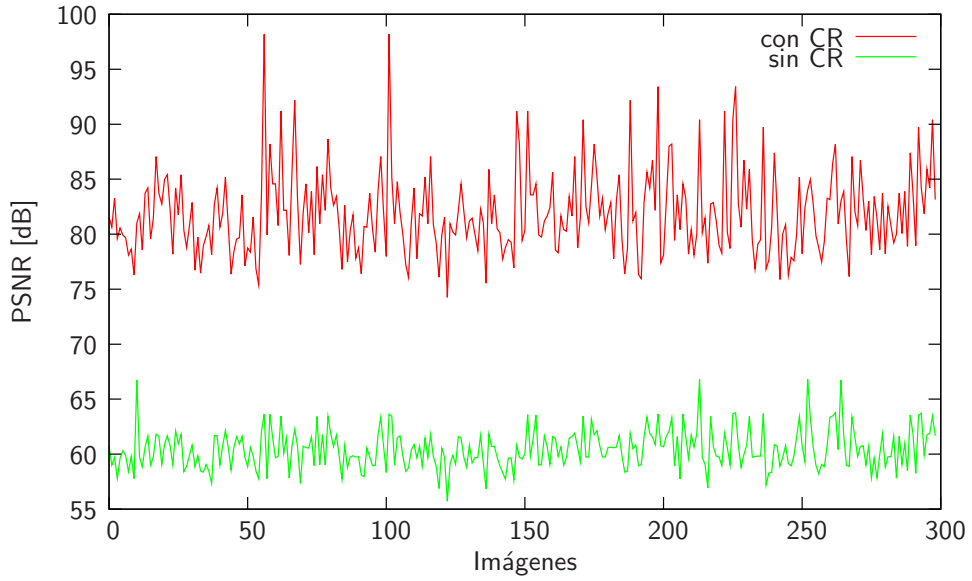


Figura 19. Gráfica que compara el PSNR obtenido para cada una de las imágenes de la secuencia *Akiyo*, cuando se utiliza una transmisión controlada por el cliente haciendo uso de CR y cuando se utiliza una transmisión estándar. El *bit-rate* empleado para la transmisión es de 11×10^6 *bits* por segundo.

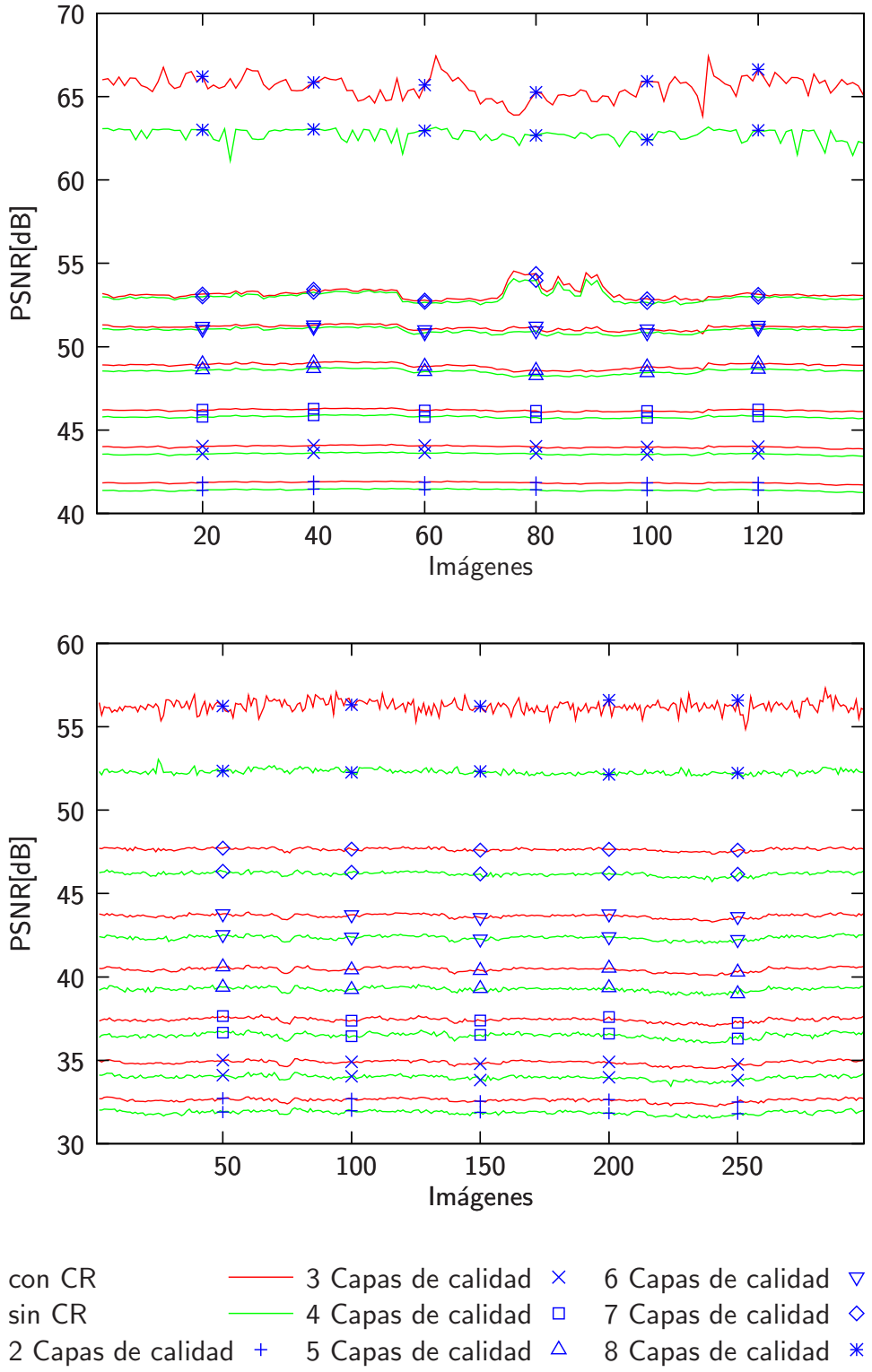


Figura 20. Comparación entre nuestra propuesta ("con CR") y el método estándar ("sin CR"). Arriba tenemos la secuencia SDO/AIA (*bit-rate* de 224×10^5 *bits* por segundo). Abajo la secuencia *Akiyo* (*bit-rate* de 8×10^5 *bits* por segundo).