University of Almería

Computer Architecture and Electronic Dept.

Interactive Browsing of Remote JPEG 2000 Images

THESIS SUBMITTED TO THE UNIVERSITY OF ALMERÍA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

Juan Pablo García Ortiz

Almería, September 2011

Directed by

Dr. Vicente González Ruiz & Dra. Inmaculada García Fernández

Abstract

The combination of the JPEG 2000 standard and the JPIP protocol is currently considered to be the state-of-art for the development of applications for remote browsing of images, specially when dealing with very large images. This combination is widely used in scientific areas (e.g. astronomy, tele-medicine, etc.).

This work has focused on these technologies, studying their strengths and providing approaches/alternatives for some of their weaknesses. The research conducted in this thesis has resulted in four different scientific and technological contributions, some of which have led to improvements to certain aspects of the technology, while others have expanded various features.

The first contribution consists of developing an architecture for interactive browsing of JPEG 2000 images based on the HTTP (HyperText Transfer Protocol), version 1.1, and the design of a new file format called JPL. The main feature of this approach is that it completely dispenses with a specific server. It only requires an easy implementation based on the use of the HTTP/1.1 and the communication is managed entirely by the clients. Experimental results show that its performance is similar to the JPIP approach.

The design of a new protocol called JPIP-W (JPIP-Web friendly) is the second contribution of this thesis. The research work has been based on the study of an effective communication protocol designed on the JPIP protocol for interactive image browsing. JPIP-W takes advantage of the Web proxy infrastructure to reduce image retrieval transmission time, user latency and network traffic. Experimental results show that JPIP-W outperforms JPIP when there is cached data. When there is no cached data, JPIP-W efficiency is virtually as good as that of JPIP.

A new efficient prefetching technique for interactive remote browsing of JPEG 2000 image sequences is also proposed in this thesis. Its most relevant characteristics are: i) it offers an easy implementation that can be added to any existing JPIP client/server architecture; ii) from the client/server bandwidth available, a certain fraction is allocated to prefetching, which is estimated using a differential quality model function; and iii) an average improvement of the reconstructed WOI is always achieved, independently

of how much fine-tuning is carried out. Experimental results prove that transitions through images of a remote sequence become smoother and also provide higher quality, improving the overall user experience.

The final contribution of this thesis work is the implementation of a new open-source and highly scalable JPIP server. This server is currently being used in the JHelioviewer project, with thousands of clients per day. The evaluation results have shown that its performance is better than the Kakadu JPIP server, in terms of scalability, and also in terms of rate-distortion.

Agradecimientos

Llegados a este punto, acabada la tesis, toca escribir los agradecimientos. Quizás sea ésta una de las partes más díficiles de escribir. No sabe uno cómo expresar de la mejor manera posible su profundo agradecimiento a todas aquellas personas que, directa o indirectamente, han estado ayudando y apoyando para que el trabajo llegue a buen puerto.

Han sido varios años los empleados, de forma intermitente, para realizar el presente trabajo. Durante este tiempo en mi camino se han ido cruzando diferentes personas que han contribuido a él de una forma u otra, dándole además forma y sentido. Estos agradecimientos están dedicados a todas ellas, pero sobre todo a las que han estado a mi lado a lo largo de todo el camino, o bien desde el principio, o bien desde siempre.

Gracias a mi amada Rosario. Sin ella este trabajo no hubiera sido en absoluto posible. Me ha acompañado tanto en lo momentos felices, con experiencias únicas en viajes por medio mundo, como en los momentos más difíciles, con noches en vela, muchos nervios, etc. El tenerla a mi lado ha sido, y sigue siendo, fundamental para mí.

Gracias a mis padres y hermana. Su cariño y apoyo incondicional han sido siempre uno de los pilares fundamentales de mi vida. Soy lo que soy, y como soy, en gran parte gracias a ellos, aportándome una rica combinación de valores y perspectivas de vida. Hoy en día ellos siguen siendo muy importantes para mí.

Gracias a los directores de mi tesis, Vicente e Inmaculada, compañeros incansables de mil y una batallas. He tenido la suerte poder contar con estos dos grandes profesionales, capaces de combinar cada uno a partes iguales el ser un gran investigador y una magnífica persona.

Gracias también a mi pequeña Tina, capaz de sacarme una sonrisa cuando más lo he necesitado.

Finalmente esperar que esto no sea el final del camino, sino una mera parada más. Tal y como diría Einstein, la vida es como montar en bicicleta, así que para mantenerse en equilibrio, hay que seguir pedaleando. Yo pretendo seguir haciéndolo, sea cual sea la dirección que tome.

Contents

CHAPTER 1

Introduction

1.1 Previous concepts

1.1.1 Digital images, compression and quality

A digital image is a rectangular matrix of digital samples taken from a bidimensional light signal. There are certain parameters that identify the image type, for example the resolution (number of samples, which is usually given/described as the width and the height of the matrix) or the bit depth (number of bits per digital sample) among others.

Each matrix item of an image is called *pixel*. Each pixel can be described by a single data item or, in general, by a set of data called components. Each component describes different properties of the pixels; for instance coloured images have three components (red, green, blue) and hyper-spectral images are usually described by more than one hundred components. The most common images have three color components (red, green and blue) per pixel, which can be described as a single data item with a bit depth of 24 bits or as three 8-bit components. In this case, the image is usually managed as three matrices with the same size, one per component, instead of only one matrix with compound items.

As can be deduced, the larger the resolution of the images, the more difficult it is to handle, store and transmit them. For example, at the moment of writing this document there are moderately-priced digital cameras capable of generating images with a resolution higher than 3000×3000 pixels. Using an uncompressed encoding method, each of these color images require a minimum of 27 megabytes. Although this volume of data per image currently does not suppose any significant drawback when stored in digital media, it does complicate the handling of the images (edition, analysis, etc.) and their transmission.

A solution to these problems is an encoding procedure known as *compression*. Compression can be defined as any technique which is able to describe the information contained in a set of data by a smaller data set. Usually, the compressed data set contains

less redundancies (redundant information) than the original one. The compression process generates a new data set from the original, smaller in size and easier to manage. The original data set can be recovered from the compressed data set by the so-called *decompression* process.

Digital images are special cases of data sets that can be compressed by eliminating not only statistical and spatial redundancy but also irrelevant information of the image which is not perceived by the human eye. In this case, when the compressed image is decompressed we do not obtain the exact original image, however, this is not noticeable by a human observer.

When the compression does not remove information, it is called *lossless* compression. When it does remove information, it is called *lossy* compression. Usually, lossy compression techniques generate data sets of smaller size than lossless compression techniques. This is why lossy compression techniques are the most commonly-used techniques nowadays.

Lossy compression techniques cause a decrease in the image quality, although visually distinguishing the original image from the compressed/decompressed image is actually very difficult for certain values of the *compression ratio*. In this type of process, the higher the percentage of lost information produced by the compression process (i.e. the higher is the compression ratio), the worse the quality of the reconstructed image, and vice versa.

The quality of a lossy compression technique is usually evaluated in terms of objective metrics which compare the original image to the reconstruction (decompressed) of the compressed image. There exist several kinds of metrics for evaluating the quality of a compressed image [24], although the most frequently used ones are the MSE (Mean Square Error) and the PSNR (Peak Signal-to-Noise Ratio).

Given an image x with a resolution of $n \times m$, a pixel located at column $i \in \{0, \ldots, n-1\}$ and row $j \in \{0, \ldots, m-1\}$ is denoted as $x[i, j]$. Let $\hat{x}$ be the image generated by a lossy compression-decompression procedure from x, then the MSE is given by

$$\text{MSE} = \frac{1}{m \cdot n} \sum_{i=0}^{(m-1)} \sum_{j=0}^{(n-1)} (x[i, j] - \hat{x}[i, j])^2. \tag{1.1}$$

The PSNR metric is more frequently used than the MSE, and it is usually expressed in dBs (Decibels). It is calculated from the MSE as

$$\text{PSNR} = 10 \log_{10} \left(\frac{(2^p - 1)^2}{\text{MSE}} \right), \tag{1.2}$$

where p refers the number of bits per component of the image.

1.1.2 Remote browsing of images

Systems for remote browsing of images are client/server architectures that allow users to explore regions of digital images that are stored in remote servers. In traditional systems, clients always have to download the complete content of the images from the server before they can make any use of them.

Alternatively, depending on the transmission mode and the compression format, clients can progressively reconstruct the images as they receive the content. At present, this kind of system is the most frequently used because of its simplicity and fast implementation.

Unfortunately, the larger the images are, in terms of physical size as well as spatial resolution, the less efficient this kind of basic system is. Furthermore, when very large images are used, clients are rarely interested in recovering the complete content at the maximum resolution but rather in recovering the full image at low resolution and a region of the image at the maximum resolution. For these cases, the idea of downloading the full file associated with the compressed image is an inefficient approach because most of the data received are not useful and frequently clients do not have the necessary bandwidth or local resources to download/manage the complete image efficiently.

There exists a large number of real world applications where users only need a part of the image saved in a remote server; e.g. astronomy, tele-microscopy [15]. For these applications, more advanced remote browsing systems are required. Another example is the tele-pathology systems based on virtual slides which are used by pathologists to remotely inspect digitalized images of human tissues. In this context, the size of the images can easily exceed 1 gigabyte [38]. Google Earth [4] is another well-known application example where users can explore images from practically any part of the world. In these cases, optimized approaches have been developed to allow the users to explore the remote images efficiently, adapting themselves to the available bandwidth and resources.

For these approaches, users interact with the application in order to define the region of the remote image they want to visualize/recover with a high level of accuracy/detail. The way this region, hereinafter called WOI (Window Of Interest), is specified will depend on the user interface and its freedom degrees. Most of the existing interfaces define this region by means of a zoom level and the spatial coordinates of the rectangular region of interest within that zoom level. In this document, WOIs will be defined by a tuple (x, y, w, h, r), where (x, y) are the coordinates of the upper-left corner of the WOI, $w \times h$ are the width and height of the region and r is the zoom level or resolution ($r = 0$ means no zoom).

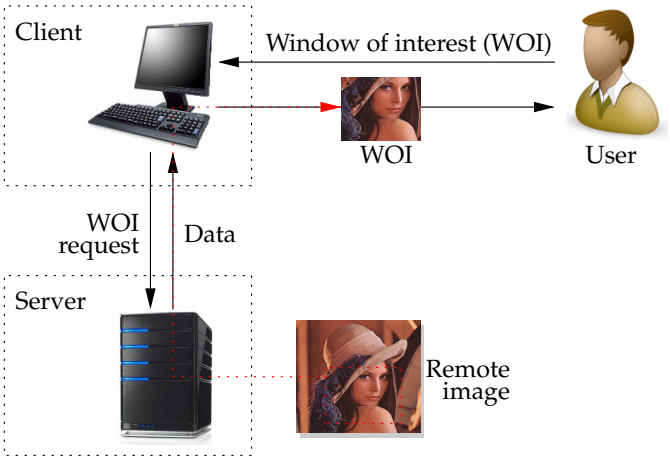

Figure 1.1: Architecture for remote browsing of images.

When the WOI is defined, the client communicates to the server sending a request. Then the server has to extract and/or process the necessary data from the associated image and sends it to the client. Only the required data to reconstruct the requested WOI are sent to the client. Therefore the client can randomly access the remote image independently of the resolution of his display and the resolution of the image. In Figure 1.1 a scheme of the structure of a remote browsing system has been depicted.

The evaluation of the remote browsing systems is mainly carried out in terms of user perception, measured as the values of the PSNR of the progressive reconstructions of the requested WOI as a function of the amount of received data. The user always wants to see an image with maximum quality as soon as possible, so this metric offers a good approach for evaluating the performance of this kind of system. Measurements of the performance of any implementation described in this thesis are based on the evolution (increase) of the PSNR with the amount of data received.

1.2 Thesis contributions and organization

The powerful features offered by the JPEG 2000 multi-part still-image compression standard (lossless/lossy compression, random access to the compressed streams, high degree of spatial and quality scalability, etc.) have led it to obtain recognition as a state-of-the-art solution among applications for remote browsing of images.

This standard, in combination with JPIP (JPeg 2000 Interactive Protocol), which is defined in its Part 9, has already been successfully used in many scientific areas (e.g. tele-microscopy [81] or tele-medicine [50]), and has a significant potential for any other area where large volumes of image data need to be streamed, like, for example, Google Earth/Maps (see Figure 1.2).

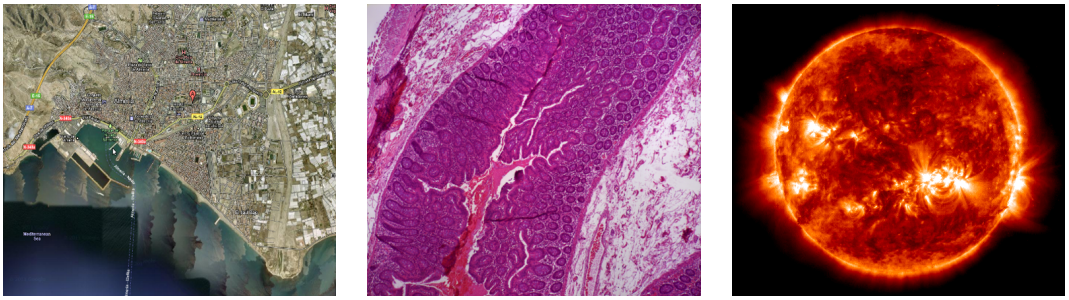

Figure 1.2: Images used in three different remote browsing systems: Google Earth (left), a tele-pathology system (center) and JHelioviewer (right).

A noticeable example in astronomy is the JHelioviewer project [61], developed by the European Space Agency (ESA) in collaboration with the National Aeronautics and Space Administration (NASA). Its main goal is to deploy a platform for the interactive data analysis and browsing. This platform should be able to accommodate the staggering data volume of 1.4 TB of images per day that will be returned by the Solar Dynamics Observatory (SDO) [65]. Among other data products, SDO will provide full-disk images of the Sun taken every 10 seconds in eight different ultraviolet spectral bands with a resolution of 4096×4096 pixels.

The main goal of this thesis has been to study the JPEG 2000 standard, as well as the JPIP, in the context of applications for interactive browsing of remote images, analyzing their strengths and providing solutions/alternatives for their weaknesses. These technologies, the JPEG 2000 standard and the JPIP, are addressed in Chapters 2 and 3, respectively.

The first main contribution of this thesis work, described in Chapter 4, has been the development of an architecture for interactive browsing of JPEG 2000 images based on the HTTP/1.1 (HyperText Transfer Protocol, version 1.1) [36, 32]. The central feature of this proposal is that it allows the implementation of efficient browsing systems, exploiting all the powerful characteristics of the JPEG 2000 standard without requiring the specific JPIP, and it also makes it possible to use any standard Web server. The proposed architecture has been written in Java, and the source code [6] has been used by several companies.

Another significant contribution of this thesis is the JPIP-W (JPIP-Web friendly), which is described and analyzed in Chapter 5 [34, 33, 35, 40]. This protocol, designed for being used over the JPIP, allows the caching system of the Web to be properly exploited, removing communication redundancies and thus improving the quality of the reconstructions.

The experience achieved studying the JPEG 2000 standard and the JPIP helped me to participate in the Google Summer of Code in 2006, developing a JPIP viewer Java

applet and a small JPIP server [28]. This code was the basis for the development of the Java viewer of the JHelioviewer project [5], which also led to my participation on said project.

As a result of this collaboration two more works were carried out and their contributions are also presented in this thesis. In Chapter 6 a novel and efficient prefetching strategy for interactive browsing of remote sequences of JPEG 2000 [39, 30] is detailed. This technique allows smooth browsing and an improved user experience.

In Chapter 7 the open-source ESA JPIP server is presented. This contribution is the result of a research project [3] which was developed through a contract between the European Space Agency and a local software company in Almeria (Spain), which also saw the collaboration of researchers from the University of Almeria. Personally, I was fortunate enough to have had the opportunity to lead all work on this research project. The performance and scalability offered by the ESA JPIP server [31] has made it the main server used in the JHelioviewer project.

During the research period of this thesis many other topics have been tackled, most of them also related to the JPEG 2000 standard like, for example, the coding and transmission of scalable video [57, 58, 56, 29], applications for tele-microscopy [70] or telepathology based on virtual slides [41, 38, 37], and client-driven conditional replenishment techniques [71], albeit they are not addressed at all in this thesis.

1.3 Acknowledgments

This work has been funded by grants from the Spanish Ministry of Science and Innovation (TIN2008-01117) and Junta de Andalucía (P08-TIC-3518), in part financed by the European Regional Development Fund (ERDF).

CHAPTER 2

The JPEG 2000 standard

2.1 Introduction

In March of 1997 the Joint Photographic Experts Group [12] opened an international call [49] looking for technical contributions in order to develop a new compression system for digital images. The goal was to conceive the successor of one of the most widespread standards: JPEG, whose name is formed by the initials of the group (Joint Photographic Experts Group) [64]. This new compression system had to solve most of the shortcomings of JPEG, as well as cover new requirements that developed in parallel to the evolution of technology.

After more than two years evaluating all of the proposals received, a modified version of the EBCOT system (Embedded Block Coding with Optimized Truncation) [77], developed by D. Taubman, was finally the winner, becoming the core of the new compression system. This core was complemented by the definition of a new file format, basic and flexible, as well as a complete data partition for profiting as much as possible from the scalability offered by EBCOT. This formed the base of the new standard, which was called JPEG 2000, defined in the first part [44]. Although this first part is the minimum required for any implementation, the standard is composed 12 Parts.

Five new parts were initially added to the standard, covering different aspects and allowing extra features. Later, in 2001, another six new parts were proposed. The main parts are described in Table 2.1. At the moment of writing this document, Parts 1-6, 8, 9, 11 and 12 were already recognized internationally as standards, Part 7 has been abandoned and Part 10 is still under development.

In this document the main parts of the standard associated with the remote browsing of images are tackled: Part 1 (and a portion of Part 2 [46]), in this chapter, and Part 9 [45], in the Chapter 3. Although there are other parts that may also be closely linked to this issue, depending on the context, like Part 8 [47], this work will only be focused on the previously-cited parts. The content of this chapter is mainly based on the JPEG 2000 book written by D. Taubman and M. Marcellin [79].

Part	Description
1	Description of the core system, which includes the base encoder/decoder and a simple image file format.
2	Extensions to the core system that introduce coding improvements, support of new data types, and a new image file format.
3	Definition and support of video files based on sequences of JPEG 2000 images.
4	Information for guaranteeing the coherence between different JPEG 2000 implementations.
5	Reference implementations of the core system: JJ2000 [25] developed in Java and JasPer [13] developed in C.
6	Definition of a new image file format for the creation of complex composed documents.
8	Extensions oriented toward security support in images and the development of JPSEC tools.
9	Description of the JPIP for the development of remote browsing systems and definition of extensions for indexing images.
10	Coding of volumetric images and support of data in floating point format.
11	Description of techniques for the detection of errors in the transmissions of JPEG 2000 images, specially in wireless communications.
12	Definition of the extensible ISO file format [43] that eases the exchange and edition of images.

Table 2.1: Parts of the JPEG 2000 standard.

When the call was opened a list of desired features that the future compression standard was required to support was also specified. JPEG 2000 not only covered all of these requirements but it also offered new and powerful features not supported by any other standard. Some of these characteristics are the following ones:

- **Wide range of bit depth:** It allows image compression with a dynamic range of bit depth for each component, from 1 bit to 32 bits. At present, JPEG 2000 is probably the only standard capable of offering this functionality.
- **High scalability:** The standard allows decompression scalability by resolution, quality and region of interest. This implies that for the same compressed image it is possible to generate reconstructions of random regions, with different resolution levels and quality levels.
- **Random access to the compressed data:** By means of several mechanisms, it is possible to randomly access the compressed context of an image. This allows, for example, the extraction of the information required for reconstructing a certain region of an image. This also makes it possible to apply simple geometric transformations without using any kind of transcoding, such as rotation (multiples of 90^o), translation and scaling.
- **Progressive transmission of images:** This feature, derived from the previous two, is essentially designed for the remote browsing of images. It allows a server to send an image in such a way that a client is able to sequentially visualize better and better approximated reconstructions of the image as it receives data. The reconstruction generated by the user depends on the data progression used. For example, with a quality progression, the client can generate a full resolution reconstruction of the original image with a quality that is incremented as the data is received from the server.
- **Lossy and lossless compression:** As will be described later, the compression process is divided into two different paths, one that is reversible and another that is irreversible. Using the reversible path, lossless compressions can be achieved.
- **Data error resilient:** JPEG 2000 defines a set of mechanisms (synchronization markers, etc.) to enable the detection of errors in the data. This feature could be useful, for example, in applications that use wireless communication channels.
- **Open architecture:** The architecture of the standard is open so that a base is defined, Part 1, to which new features can be incorporated depending on new specific requirements.

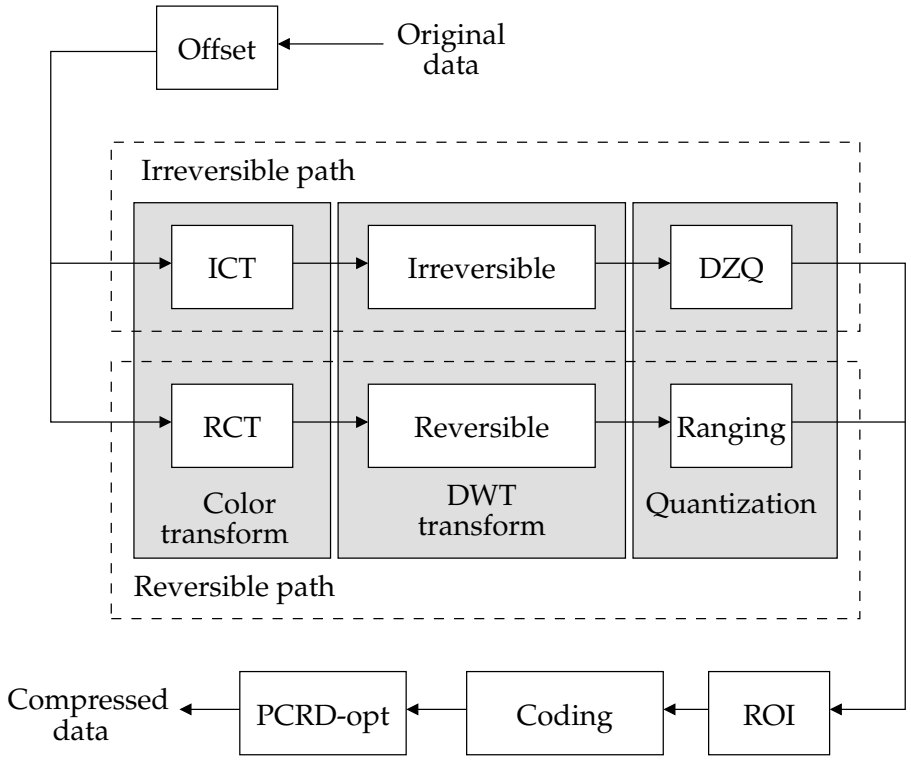

Figure 2.1: Architecture of the JPEG 2000 core coding system.

- **Better performance with low bit-rates:** At low bit-rates, the performance offered by JPEG 2000 is better than other standards. This property is specially interesting in applications for remote browsing of images.

- **Possibility of defining regions of interest (ROI):** In practice, a user can be interested in certain parts of an image which are more important than others for a specific task. The standard makes it possible to define certain ROIs within an image with the aim of being coded and transmitted with a higher priority than the rest. Notice that the term ROI used by the standard is not the same as the term WOI used in this document. ROI and WOI refer to different concepts.

2.2 The core coding system

Part 1 of the standard defines the basic architecture of the compression system, formed by a set of processing stages, which has been depicted as a block diagram in Figure 2.1. As can be observed in this figure, the compression stages are divided into two different paths, one that is irreversible and another that is reversible, depending on the kind of compression, lossy and lossless, respectively. Lossy compression, the most frequently used, achieves the highest compression ratio at the expense of a slight loss in image quality

The processing stages (ICT, DWT, DZQ, etc.) are applied sequentially and independently to each image *tile*. In the next section this and other terms that are defined by the standard for the data partition are explained.

Next, all the processing stages that are part of the two different paths of the compression process are explained:

- **Offset:** Each sample i of the image to compress, which is called $x_i[\boldsymbol{n}]$, where $\boldsymbol{n}$ refers to the coordinate points $[n_1, n_2]$, with a depth of B bits, must be signed values in the range
$$-2^{B-1} \leq x_i[\boldsymbol{n}] \leq 2^{B-1}.$$
Therefore, if the image source samples are unsigned, which is the most frequent case, an offset of -2^{B-1} has to be added.

- **Color transform (ICT, RCT):** The color transform is optional, since it can only be applied when there are at least three color components and bit depth. It is assumed that these first three components are red, green and blue (RGB).

  The goal of the color transform is to represent the RGB components in a different color model, which allows the system to reduce the existing color redundancy. The compression ratio is increased with this operation.

  As can be appreciated in Figure 2.1, there are two kinds of color transforms, one for the irreversible path, ICT (Irreversible Color Transform), and another one for the reversible path, RCT (Reversible Color Transform). The ICT transforms from RGB to YCbCr. The RCT transforms from RGB to Y'DbDr.

  Assuming a sample with components red $x_{\mathrm{R}}[\boldsymbol{n}]$, green $x_{\mathrm{G}}[\boldsymbol{n}]$ and blue $x_{\mathrm{B}}[\boldsymbol{n}]$, the ICT is defined as
$$x_{\mathrm{Y}}[\boldsymbol{n}] = \alpha_{\mathrm{R}} x_{\mathrm{R}}[\boldsymbol{n}] + \alpha_{\mathrm{G}} x_{\mathrm{G}}[\boldsymbol{n}] + \alpha_{\mathrm{B}} x_{\mathrm{B}}[\boldsymbol{n}], \tag{2.1}$$
$$x_{\mathrm{Cb}}[\boldsymbol{n}] = \frac{0.5}{1-\alpha_{\mathrm{B}}}(x_{\mathrm{B}}[\boldsymbol{n}] - x_{\mathrm{Y}}[\boldsymbol{n}]), \tag{2.2}$$
  and
$$x_{\mathrm{Cr}}[\boldsymbol{n}] = \frac{0.5}{1-\alpha_{\mathrm{R}}}(x_{\mathrm{R}}[\boldsymbol{n}] - x_{\mathrm{Y}}[\boldsymbol{n}]), \tag{2.3}$$
  where $\alpha_{\mathrm{R}} = 0.2999$, $\alpha_{\mathrm{G}} = 0.587$ y $\alpha_{\mathrm{B}} = 0.114$. Alternatively for the same components, the RCT is defined according to
$$x_{\mathrm{Y'}}[\boldsymbol{n}] = \left\lfloor \frac{x_{\mathrm{R}}[\boldsymbol{n}] + 2x_{\mathrm{G}}[\boldsymbol{n}] + x_{\mathrm{B}}[\boldsymbol{n}]}{4} \right\rfloor, \tag{2.4}$$
$$x_{\mathrm{Db}} = x_{\mathrm{B}}[\boldsymbol{n}] - x_{\mathrm{G}}[\boldsymbol{n}] \tag{2.5}$$

and

$$x_{\mathrm{Dr}} = x_{\mathrm{R}}[\boldsymbol{n}] - x_{\mathrm{G}}[\boldsymbol{n}]. \tag{2.6}$$

All the images used in this work meet the condition required for applying the color transform as they are color images with three RGB components.

- **Wavelet transform:** The third stage is based on the Discrete Wavelet Transform (DWT). This makes it possible to represent the samples of the image to be compressed in a dual spatial-frequency domain that has three important advantages:

  1. Normally, the high-order entropy of the image is smaller in the DWT domain than that of the image, and,therefore, the lossless compression of the signal needs less memory if it is performed in the transformed domain.
  2. Thanks to the typical decorrelation effect produced by the DWT, most of the energy of the natural[1] images is concentrated into a small set of low-frequency elements.This allows for the design of lossy image compressors based on low-pass-filtering and/or quantization processes.
  3. Each component is transformed into a multi-resolution representation, which is very beneficial in visualizing and handling tasks.

The DWT can be described from the point of view of the Filter Bank Theory [73]. According to this theory, a one-dimensional sequence of N samples

$$S = \{s[n];\ n = 0, \cdots, N-1\}$$

can be represented by means of two half size sequences:

$$L = \{l[n];\ n = 0, \cdots, \frac{N}{2}\}$$

and

$$H = \{h[n];\ n = 0, \cdots, \frac{N}{2} - 1\},$$

where L is the result of filtering S using a low-pass filter, and H is the result of filtering S using a high-pass filter. From the point of view of the frequency domain, this bank of two filters is defined in such a way that the low-pass filter does not eliminate what the high-pass filter cuts, and vice versa. That is the reason why all of the information contained in bands L and H is the same as that in the source sequence S.

This decomposition process of one signal into two, one with the low frequencies and another one with the high frequencies, can be applied recursively to both sub-

[1] Resulting from digitalizing natural scenes.

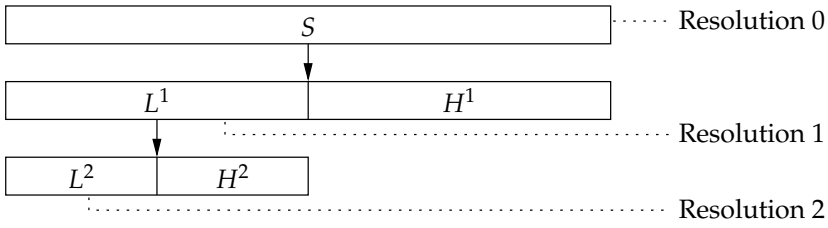

Figure 2.2: Dyadic DWT decomposition of 2 levels.

bands. When it is only applied to the low-frequency band, the DWT is called dyadic. This version is the most frequently used for image compression because most of the energy is accumulated in the low-frequency bands.

Figure 2.2 shows the decomposition process for the one-dimensional dyadic DWT. Specifically, in this figure two different decomposition stages, which generate three resolution levels, are shown. As can be observed, for R resolution levels, $R-1$ decomposition stages are required, and bands $L^{R-1}[n]$, $H^{R-1}[n]$, $H^{R-2}[n]$, $\cdots$, $H^1[n]$ are obtained, with $L^0 = S$ and H^0 being the zero-energy signal. The values of the resulting sub-bands are called *wavelet coefficients*.

Unlike other transforms such as the DFT (Discrete Fourier Transform) or the DCT (Discrete Cosine Transform), the DWT defines a representation not only in the frequency domain, but also in the spatial one. This property allows the extraction of a subset of wavelet coefficients in order to only reconstruct a part of the original signal at a lower resolution level.

The multidimensional DWT is separable, so it is possible to calculate the two-dimensional (2D) DWT of an image just by first applying the one-dimensional DWT to each row of the image and then to each column of the resulting image (or vice versa). Thus, after each decomposition stage four bands (LL, LH, HL, and HH) are obtained instead of two.

In Figure 2.3 the result of applying two decomposition stages of 2D-DWT to an image of Lena is shown. In this case, $3R-2$ sub-bands are generated for R resolution levels. Each sub-band is identified by a subindex which refers to the associated decomposition stage, and a couple of identifiers - L and H (low-pass and high-pass filter, respectively), indicating which filter was applied first.

As can be seen in Figure 2.3, in order to recover a $X/2^R \times Y/2^R$-resolution version of the full $(X \times Y)$-resolution image after R iterations of the 2D-DWT, only the LL^R sub-band is necessary. For recovering the image at the next incremental resolution level, $Y/2^{R-1} \times X/2^{R-1}$, the sub-bands HL^R, LH^R and HH^R are necessary, as well as others. This means that after a R-levels 2D-DWT, different $R+1$ resolutions of the image have to be reconstructed. For this reason, the DWT contributes to the resolution scalability

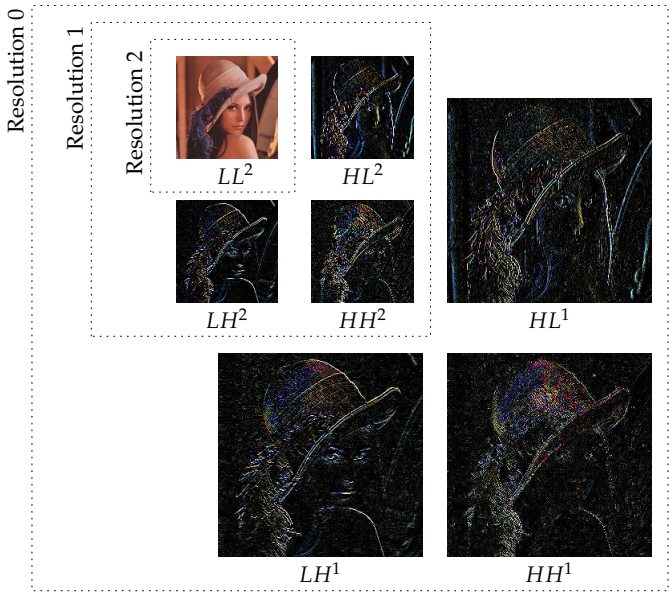

Figure 2.3: Two-dimensional decomposition of the dyadic DWT, $R = 3$.

of the JPEG 2000 standard. A decompressor can reconstruct versions of the image at a lower resolution level using only a certain subset of the wavelet coefficients, without requiring any additional processing.

- **Quantization:**

  The quantization is the operation that maps each coefficient ω, provided by the DWT, to an index using the expression

  $$q_b(\omega) = sign\,(\omega) \left\lfloor \frac{|\omega|}{\Delta_b} \right\rfloor . \tag{2.7}$$

  The JPEG 2000 standard permits the definition of a quantization step-size, Δ_b, specific of each sub-band b. For the reversible path, the quantization step-size must be equals 1. For the irreversible path, the step-size is obtained in term of a mantissa μ_b of 11 bits and an exponent ε_b of 5 bits, according to

  $$\Delta_b = 2^{-\varepsilon_b} \left(1 + \frac{\mu_b}{2^{11}}\right). \tag{2.8}$$

  Through the irreversible path the quantization is called *Dead Zone Quantization*, and through the reversible path *Ranging*.

- **Definition of the ROIs (Regions Of Interest):**

  JPEG 2000 offers mechanisms that can be used by a compressor in order to assign a higher priority (for the rate control) to certain regions of an image. These

regions, that can adopt any geometric shape, are decoded with a higher quality than the rest of the image.

The most frequently used method for specifying a ROI in the standard JPEG 2000 is Maxshift. It consists of applying an offset of s bits to the wavelet coefficients of the ROI, referred to as $q(\omega)$. This offset is applied by means of

$$q'(\omega) = q(\omega)2^s. \tag{2.9}$$

- **Coding:**

  The coding is carried out using a bit plane coder based on the EBCOT and an arithmetic code called MQ. MQ is a version of the Q coder [72] adopted in the standard JBIG [48].

  As a result of the coding process, a new data sequence, also called bit-stream, is generated. This coding process removes the statistical redundancy of the bit planes of the wavelet coefficients.

  The wavelet coefficients are grouped into rectangular regions with a fixed size (commonly 32×32 or 64×64) that are called *code-blocks*. The coding process is applied independently to each code-block. For each code-block an independent compressed bit-stream is obtained. This way of coding contributes to the spatial scalability of the standard. Therefore, a decompressor can reconstruct a certain region of the image just by decompressing the associated code-blocks.

  Thanks to the EBCOT procedure, another method for the ROI definition can be used. The larger the number of coefficients of a code-block associated to a certain ROI, the higher the priority given to that code-block for coding. The advantage of this method is that it allows the redefinition of a ROI by simply modifying the ordering of the decoding of the code-blocks. The drawback is that the definition of the ROI is limited by the borders of the code-blocks.

- **PCRD-opt:**

  The PCRD-opt algorithm (Post Compression Rate-Distortion optimization) is used for finding the optimal segmentation for the quality layers. In this last stage of the JPEG 2000 encoding process, the bit-stream resulting from the coding of each code-block is divided into N contiguous segments, which are called *quality layers*. These are created so that if only the first n segments of the code-stream are decompressed, the optimal quality is always achieved. Each code-block i generates a compressed bit-stream B_i of size L_i. The segmentation of this stage affects

each code-block in such a way that generating N segments of the total bit-stream is equivalent to generating N segments of all the code-block bit-streams.

Let $L_i^{(n)}$ be the length of segment n of code-block i. Thus, the total length of the quality layer is

$$L_n = \sum_i L_i^{(n)}, \tag{2.10}$$

and each segment $L_i^{(n)}$ of each code-block i contributes to decreasing the distortion of the reconstruction ($D_i^{(n)}$), measured in terms of the MSE or the PSNR, and therefore, to increasing image quality. Since the distortion is an additive measurement, the total distortion that segment n of the code-stream produces on the quality of the image is:

$$D^{(n)} = \sum_i D_i^{(n)}. \tag{2.11}$$

The PCRD-opt algorithm [77] is basically an optimization technique that tries to minimize the total distortion for each n (Equation 2.11) with the restriction $L^{(n)} \leq K^{(n)}$, where $K^{(n)}$ is an established upper bound of the length for each quality layer. The ideal combination of subsegments of code-blocks is chosen for obtaining the minimum total distortion of the image.

Notice that the PCRD-opt stage contributes to the quality scalability of the standard. The decompressor can generate reconstructions with lower quality using only a certain set of segments which includes the initial segments of each code-block.

2.3 Data partitions

The JPEG 2000 standard defines a wide variety of partitions for the image data, with the aim of exploiting the scalability offered as much as possible. These partitions are intended to efficiently manipulate the full image or a piece of the full image. Figure 2.4 shows a graphical example of the main data partitions.

In order to understand the concept of each partition defined in the JPEG 2000 standard, it is necessary to clarify the concept of *canvas*. The canvas is a bidimensional drawing zone where all the partitions are mapped to form the corresponding image. Hereinafter, all the used coordinates are described with respect to a canvas, whose size, width and height correspond to the total size of the associated image. This does not necessarily imply that the compressed image must occupy the entire the canvas. Each partition is located and mapped over the canvas in a specific way.

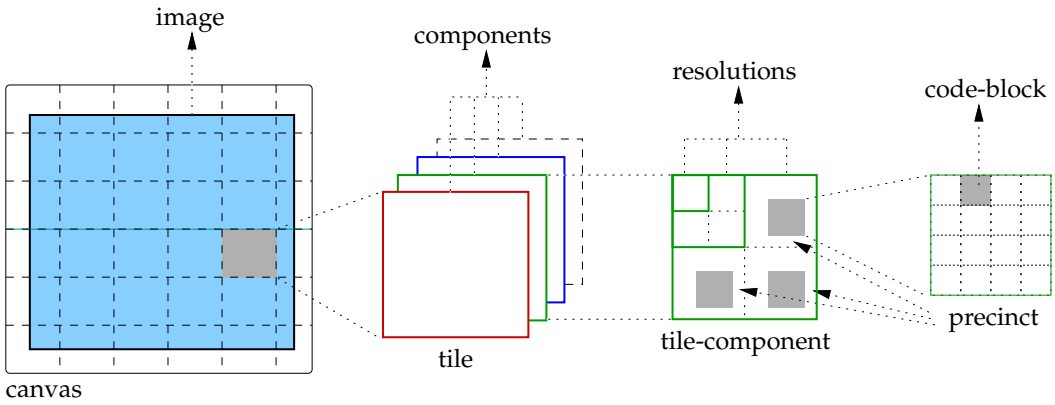

Figure 2.4: Data partition defined by the JPEG 2000 standard.

An image is composed by one or more components. In most of the cases the images have only three components: red, green and blue (RGB). In the JPEG 2000 standard the components have associated sub-sampling factors. Having a component c, defined by a bi-dimensional set of samples, $x_c[n_1, n_2]$, two sub-sampling factors are defined, S_1^c and S_2^c for the rows and the columns, respectively. Each sample of the component, $x_c[n_1, n_2]$ has a position $[n_1 S_1^c, n_2 S_2^c]$ within the canvas. For a simple RGB image, the three color components have the same size as the canvas, and sub-sampling factors equal to 1.

The JPEG 2000 standard allows an image to be divided into smaller rectangular regions called *tiles*. Each tile is compressed independently with respect to the rest, meaning the compression parameters can be different for each tile. The tile partition is defined by means of four positive integer parameters: the first two, T_1, T_2, define the height and width of the tiles, and the last two Ω_1^T and Ω_2^T, define the anchor point of the tiles (see Figure 2.5).

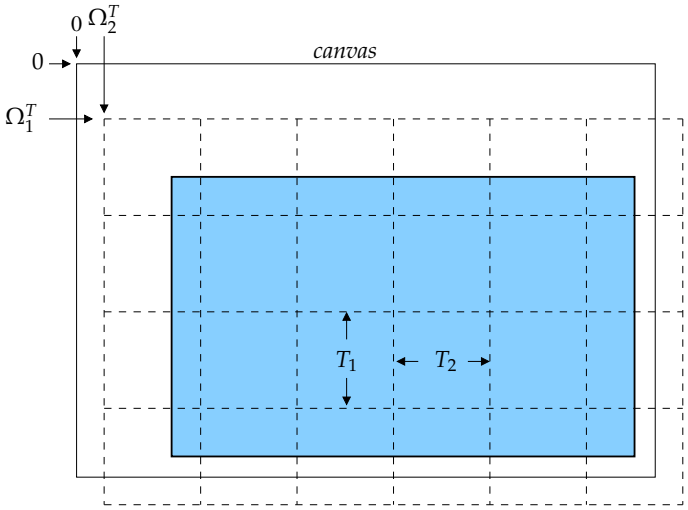

Figure 2.5: Tiling partition over a canvas. The blue region refers to the image, and the rectangles with discontinuous lines are the tiles.

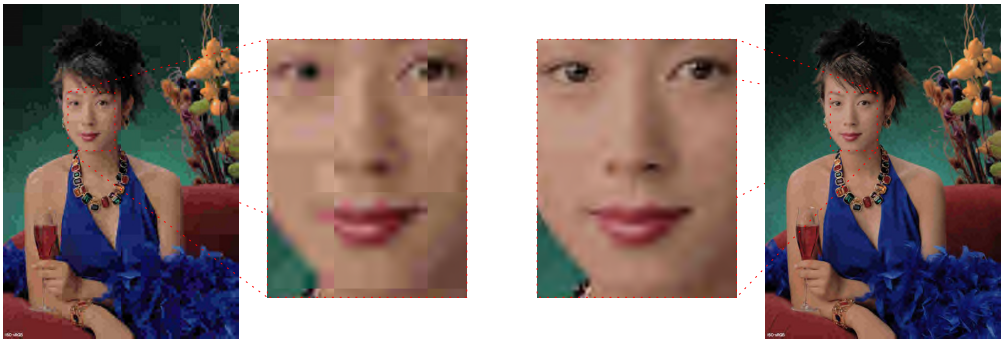

Figure 2.6: The image "Woman with glass" of the standard ISO 12640-2 compressed with 0.01 bpp, with tiles of 256 × 256 (left) and without tiles (right).

One of the possible applications of these partitions is their use with images that contain several elements which are visually different and separated, such as text, graphics or photographic materials. If this is not the case and the images are continuous and homogeneous, tiling is not advisable because it produces artifacts on the borders of the tiles, causing a mosaic effect. An example of the tiling effect in images can be seen in Figure 2.6. In one image the tile borders can easily be perceived. Moreover, when the tiles are used, the size of the compressed image is larger.

The DWT and all the quantization/coding stages are independently applied to each *tile-component*. A tile-component, of a tile t and a component c, is defined by the two-dimensional rectangular region covered by t, taking into account the sub-sampling factors of c[2]. This means that if an image has only one tile with three color components, there are three tile-components which are compressed independently.

For each tile-component, identified by tile t and component c, there are a total of $R_{t,c}$ resolution levels. The r-th resolution level of a compressed tile-component is obtained after applying $(R_{t,c} - r)$ times the inverse DWT. The value of r is in the range of $0 \leq r < R_{t,c}$. The biggest resolution, $r = 0$, refers to the source tile-component. In Section 2.2, an example of the partition generated by the DWT is explained in more detail.

A certain resolution level r of the complete image is obtained generating the resolution level r of all the associated tile-components. The standard allows each tile-component to have a different number of resolution levels.

Let $[E_1^t, F_1^t) \times [E_2^t, F_2^t)$ be the region occupied by the tile t over the canvas. The size of the tile-component for tile t and component c, within resolution r, is defined by

$$E_i^{t,c,r} = \left\lceil \frac{E_i^t}{2^r S_i^c} \right\rceil, F_i^{t,c,r} = \left\lceil \frac{F_i^t}{2^r S_i^c} \right\rceil, \text{ for } i \in \{1,2\}. \tag{2.12}$$

[2]In lossy coding, the RGB domain is transformed to the Y'DbDr domain and the DbDr components are typically sub-sampled.

Each tile-component in the DWT domain is divided into code-blocks which are then independently coded. This code-blocks partition is defined by the anchor point $[\Omega_1^C, \Omega_2^C]$, and the maximum height and width of the code-blocks $J_1^{t,c}$ and $J_2^{t,c}$. In Part 1 of the standard, the anchor point must obligatorily be $[0, 0]$ but, on the contrary, in Part 2 each $\Omega_i^{t,c}$ can be either 0 or 1. The sizes commonly used for the code-blocks are 32×32 and 64×64, although the size can be

$$J_i^{t,c} = 4 \cdot 2^{E_i}, \text{ for } i \in \{1, 2\}, \tag{2.13}$$

subject to

$$\begin{aligned} 0 \leq E_i \leq 8, \\ 0 \leq (E_1 + E_2) \leq 8. \end{aligned} \tag{2.14}$$

For each resolution r of each tile-component (t, c), the code-blocks are grouped into *precincts*. This partition is defined by the height and width of each precinct, given in term of the number of code-blocks, $P_1^{t,c,r}$ and $P_2^{t,c,r}$, respectively.

Each precinct P is identified by a pair of indexes $[p_1, p_2]$, which satisfy the restrictions

$$\begin{aligned} 0 \leq p_1 < N_1^{P,t,c,r}, \\ 0 \leq p_2 < N_2^{P,t,c,r}, \end{aligned} \tag{2.15}$$

where $N_1^{P,t,c,r}$ and $N_2^{P,t,c,r}$ identify the number of precincts vertically and horizontally, respectively. These values are obtained by means of

$$N_i^{P,t,c,r} = \begin{cases} \left\lceil \frac{F_i^{t,c,r} - \Omega_i^C}{P_i^{t,c,r}} \right\rceil - \left\lfloor \frac{E_i^{t,c,r} - \Omega_i^C}{P_i^{t,c,r}} \right\rfloor & \text{if} \quad F_i^{t,c,r} > E_i^{t,c,r}, \\ 0 & \text{if} \quad F_i^{t,c,r} = E_i^{t,c,r}. \end{cases} \tag{2.16}$$

Code-blocks refer to the wavelet coefficients generated by the DWT, which are grouped into rectangular regions within the wavelet domain. However, precincts refer to rectangular regions within the image domain. This means that a precinct of resolution r includes code-blocks from the sub-bands LH^r, HL^r and HH^r, which are related to the same rectangular region represented by the precinct. When $r = 0$, the precinct includes only code-blocks from the sub-band $LL^{(R^{t,c}-1)}$.

The *packet* is the fundamental unit for the organization of the compressed bit-stream of an image. Each precinct contributes to the bit-stream with as many packets as quality layers there are.

The compressed data of each code-block is composed of several different segments with a variable length; there are as many segments as quality layers. All the code-blocks of all the precincts of the same tile are divided into the same number of quality

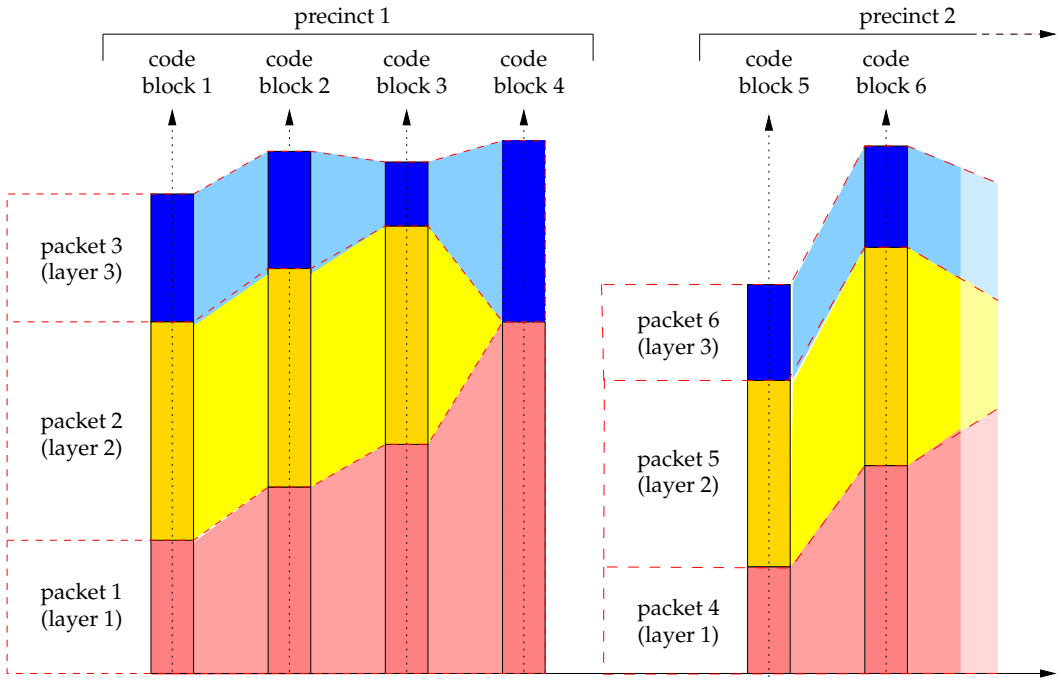

Figure 2.7: Example of quality layers and packets.

layers, although the length of the quality layers for each code-block can be different (the length can even be zero). For a certain layer l, the set of the entire layer l of all the code-blocks related to a precinct form a packet. Figure 2.7 shows a graphical example.

The number of quality layers of a tile t is identified as Λ_t. Although the number of quality layers can vary from tile to tile, compressors commonly use the same number of layers for all the tiles in order to avoid ambiguities when it is required to discard a certain number of quality layers for an image.

The total number of packets generated after the compression of a certain image is given by

$$\sum_{t_1=0}^{N_1^T-1}\sum_{t_2=0}^{N_2^T-1}\sum_{c=0}^{C-1}\sum_{r=0}^{R_{t,c}-1}\Lambda_t N_1^{P,t,c,r} N_2^{P,t,c,r}. \tag{2.17}$$

A packet $\zeta_{t,c,r,\boldsymbol{p},l}$ is identified by the tile t, the component c, the resolution r, the precinct $\boldsymbol{p}$ ($\boldsymbol{p} \equiv [p_1, p_2]$) and the quality layer l.

2.4 Code-stream organization

Part 1 of the JPEG 2000 standard defines a basic structure for organizing the image compressed data into code-streams. A code-stream includes all the packets generated by a compression process of an image plus a set of *markers*. Markers are used for signaling certain parts, as well as for including information necessary for the decompression.

The code-stream is itself a simple file format for JPEG 2000 images. Any standard

decompressor must be able to understand a code-stream stored within a file. This basic format is also called raw, and its most frequently used extension is ".J2C".

Markers have a unique identifier, that consists of an unsigned integer of 16 bits. These markers can be found alone, that is, only the identifier, or accompanied by additional information, receiving in this case the name of *marker segments*.

After the identifier, the marker segment has another unsigned integer of 16 bits with the length of the data including the two bytes of this integer, but without counting the two bytes of the identifier.

All the available markers in Part 1 of the standard can be examined in Table 2.2. Not all the markers are explained in detail in this document, but rather only the minimum set of markers required for any code-stream, and those related to this work.

The code-stream always begins with the SOC (Start Of Code-stream) marker, which does not include any additional information. After this marker, a set of markers called "main header" begins. The markers that can appear within this main header are enumerated in Table 2.2.

The first marker of the main header that appears just after the SOC is SIZ, with global information necessary for decompressing the data, e.g. the image size, the tile size, the anchor point of the tiles, the number of components, the sub-sampling factors, etc.

There are another two markers that are mandatory in the main header: i) COD, with information related to the coding of the image (like the number of layers, number of DWT stages, the size of the code-blocks, the progression, etc.) and ii) QCD, which contains the quantization parameters. These two markers can be stored in any position within the main header.

The rest of the code-stream, until the EOC (End Of Code-stream), located just at the end, is organized as shown in Figure 2.8. For each image tile, there is a set of data. This data is divided into one or more *tile-parts*. Each tile-part is composed by a header and a set of packets. The header of the first tile-part is the main header of the tile. The header of each tile-part begins with the SOT (Start Of Tile) marker and ends with the SOD (Start Of Data) marker, followed by the related sequence of packets, according to the last COD or POC marker. The main header ends when the first SOT is found.

In order to permit random access to the data of a code-stream, that by default is not feasible, JPEG 2000 offers the possibility of including the TLM, PLM and/or PLT markers. The TLM and PLM markers are included within the main header, whilst the PLT marker goes into the header of a tile or tile-part. The goal of the TLM marker is to store the length of each tile-part that appears within the code-stream. This length includes the header as well as the set of packets; so, in order to know where the beginning of

Name	Identifier	Description	Position	Data?	Required?
SOC	0xFF4F	Start of the code-stream	M	No	Yes
SOT	0xFF90	Start of a tile or tile-part	T,P	Yes	Yes
SOD	0xFF93	Start of the data of a tile or tile-part	T,P	No	Yes
EOC	0xFFD9	End of code-stream	E	No	Yes
SIZ	0xFF51	Size of the image and tiles	M	Yes	Yes
COD	0xFF52	Coding parameters	M,T	Yes	Yes
COC	0xFF53	Component coding parameters	M,T	Yes	No
QCD	0xFF5C	Quantization parameters	M,T	Yes	Yes
QCC	0xFF5D	Component quantization parameters	M,T	Yes	No
RGN	0xFF5E	Region of interest (ROI)	M,T	Yes	No
POC	0xFF5F	Progression order change	M,T,P	Yes	No
TLM	0xFF55	Lengths of all the tile-parts of the code-stream	M	Yes	No
PLM	0xFF57	Lengths of all the packets of the code-stream	M	Yes	No
PLT	0xFF58	Lengths of all the packets of a tile-part	T,P	Yes	No
PPM	0xFF60	Headers of all the packets of the code-stream	M	Yes	No
PPT	0xFF61	Headers of all the packets of a tile-part	T,P	Yes	No
SOP	0xFF91	Starting of a packet	S	Yes	No
EPH	0xFF92	End of a packet header	S	Yes	No
CRG	0xFF63	Component register	M	Yes	No
COM	0xFF64	Comment	M,T,P	Yes	No

Table 2.2: Markers of a JPEG 2000 code-stream. The positions where the markers can be located are identified by: M, for the main header; T for the header of a tile; P, for the header of a tile-part; S, when the marker appears within the compressed data of a tile-part; E: when it appears at the end of the code-stream.

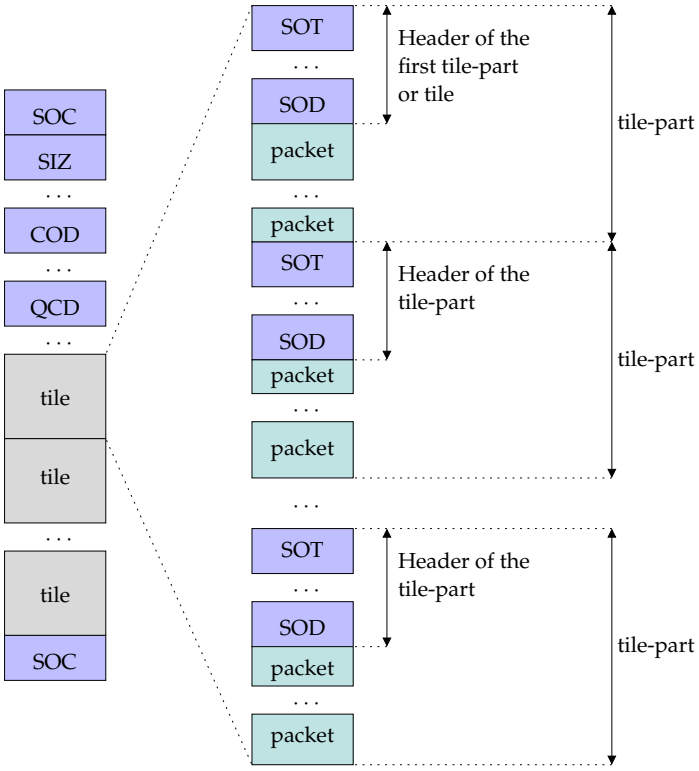

Figure 2.8: Code-stream organization.

the data is, it is necessary to first analyze the header. The PLM marker stores the length of each packet of each tile-part of the code-stream. Each packet of the code-stream has a certain length, which is a priory unknown. Therefore including this marker facilitates random access to the packets. The PLT marker has the same function as the PLM marker, but at the level of tile-part, thus it stores the length of all the packets of the tile-part it belongs to.

Obviously, the PLM and PLT markers produce an increase of the code-stream length, although the way of coding the length of the packets helps to avoid an excessive overhead [3]. The most significant bit of each byte indicates if the byte is (1) or is not (0) the last one of the sequence. This way of encoding is widely used in Part 9 of the standard, specially with the JPIP. With this protocol, each sequence of bytes that represents a number encoded in this way is called VBAS (Variable Byte-Aligned Segment).

The SOP and EPH markers can appear interleaved with the packets. If the SOP marker is used, it always appears at the beginning of each packet and can be used to detect possible errors in the sequence: for example, in the PLM and PLT markers the packet lengths include the length of the SOP marker as well, so if these markers

[3] A length L of a certain packet, that can be represented with B_L bits, is stored coded with $\left\lceil \frac{B_L}{7} \right\rceil$ bytes.

are used to randomly access the packets, examining the first two bytes read from the packet, it is possible to check whether the data is correct or not (these two first bytes must be equal to the identifier of the SOP marker). The SOP marker stores the index of the associated packet as an unsigned 2-bytes integer. If the EPH marker is used, it appears at the end of each packet and it does not contain any useful information. Its function is only to delimit a packet. Within the COD marker, it is specified if the SOP and EPH markers are used.

2.5 Progressions

The packets generated by the JPEG 2000 compression process are neither independent nor self-contained. Having a certain packet without additional information, it is not possible to figure out to which part of the related image it belongs. The length of the packet cannot be determined before being decoded, and many packets cannot be decoded without decoding other packets prior. This is why it is necessary to include markers like TLM, PLT or PLM, as previously commented, in order to allow random access.

When the data access is sequential, these markers are not required, but the order in which the packets are stored must be established. Therefore, the decompressor, with knowledge of this order beforehand, can identify the packets and their relations with the image as they are being decoded. This order is called *progression*. In JPEG 2000 the packets must always follow a specific progression, either using markers for random access or not. The packets of each tile-part appear according the progression specified by the last COD or POC marker read before the SOD marker.

Part 1 of the JPEG 2000 standard defines 5 possible kinds of progressions for ordering the packets within a tile or tile-part. Each progression is identified by means of a combination of four letters: "L" for quality layer, "R" for resolution level, "C" for component and "P" for precinct. Each letter identifies the partition of the progression. Thus, for the LRCP progression, for example, the packets would be included as follows:

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

The resolution levels are covered from the smallest to the biggest. The different

progressions allowed by the standard are enumerated and commented on below:

1. **LRCP progression:**

   This is a progression in quality, so that a new layer of a tile is not decoded until all the packets of the previous layers have been decoded.

2. **RLCP progression:**

   It is a progression in resolution, since the packets are decoded from one resolution to the next, beginning with the smallest one, including all the quality layers.

3. **RPCL progression:**

   Similarly to the previous one, this is also a progression in resolution. The difference is, that for each resolution, the packets are decoded precinct by precinct, whilst in the previous one they are decoded layer by layer.

4. **PCRL progression:**

   This is a progression in position, so that the packets are decoded precinct by precinct, beginning with the one located in the upper-left border, and ending with the bottom-right border.

5. **CPRL progression:**

   The last progression offered by the standard is a progression in component. All the packets of a component are decoded only after decoding all the packets of the previous component.

The selection of a progression or another depends on the application to be carried out and how the packet must be decoded. For example, if the packets are going to be accessed randomly, but minimizing the disk access as much as possible is required, RPCL would be the ideal progression in this case. With this progression the packets are mainly distributed by resolution, and the fact of having "L" in the last place means that the packets of each precinct are contiguous.

In the case of image transmission, the packets must also follow a specific order or progression when they are transmitted. This progression may be different from those defined by the standard, although this would hinder any implementation.

When an image is transmitted from a server to a client, the most desirable goal is to allow the client to be able to show reconstructions of the image with the highest quality possible according to the amount of data received. Under this criteria, the LRCP progression is confirmed to be the best.

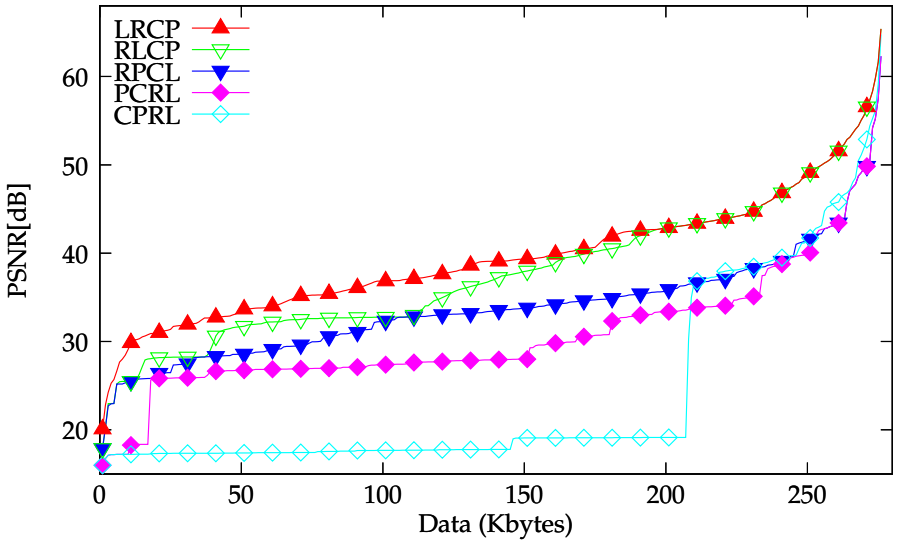

Figure 2.9: Comparison of the progression orders using the Lena image.

Figure 2.9 shows the result of a comparison between the different JPEG 2000 progressions decoding Lena's image. The horizontal axis refers to the amount of sequential data, in kilobytes, that is decoded from the compressed image. The vertical axis refers to the quality, in PSNR[dB], of the reconstruction obtained using this amount of data. As can be seen, the LRCP progression is the one that achieves the highest quality of the image reconstructions for any amount of data received.

2.6 File formats

Although the code-stream is completely functional as a basic file format, it does not permit the inclusion of additional information that could be necessary in certain applications, e.g. meta-data, copyright information, or color palettes. By means of the COM marker, auxiliary information can be included within a code-stream, but it is not classified nor organized in a standard way.

Part 1 of the standard also defines a file format based on "boxes" that makes it possible to include, for example, in the same file, several code-streams and diverse information that is correctly identified. These files usually have the extension ".JP2"; extension is also used for identifying this kind of file.

The JP2 files are easily extensible. A basic structure of a box is defined as something that can contain any kind of information. Each box is unequivocally classified by means of a 4-bytes identifier. The standard proposes an initial set of boxes, that may be extended according to specific requirements. In fact, the JP2 format is the basis for all other formats and extensions defined in the remaining parts of the standard.

The list of boxes defined for a JP2 file, as well as its organization, are shown in

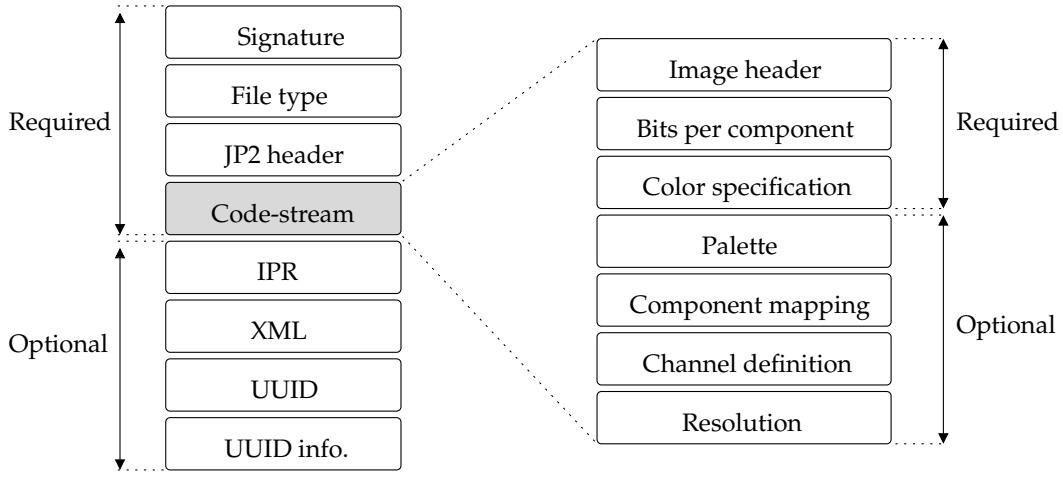

Figure 2.10: Organization of a JP2 file.

Figure 2.10. Each box has a header of 8 bytes. The first 4 bytes, L, form an unsigned integer with the length (in bytes) of the content of the next 4 bytes, while T contains the identifier of the kind of box. This identifier is commonly treated as a string of 4 ASCII characters. The value of L includes the header, thus the real length of the content of the box is $L - 8$. L can have any value bigger or equal to 8, but also 1 or 0. If $L = 1$ the length of the content of the box is coded as an unsigned integer of 8 bytes, X, located after T. In this case the header occupies 16 bytes and the length of the content is then $X - 16$. If $L = 0$ the length of the box content is undefined, which is only possible in the case of the last box of the image file.

Boxes can contain other boxes. It is possible to know whether a box contains sub-boxes or not depending on the value of T. If a box contains sub-boxes, it can only contain sub-boxes, so it cannot combine sub-boxes with other data.

As in the case of the code-stream markers, there are some boxes which must always be present in a JP2 file. These boxes are:

1. The "JPEG 2000 signature" box, which must be unique. Its goal is to mark the beginning of the file in order to detect possible transmission errors.

2. The "File type" box that contains the list of formats the file belongs to.

3. The "JP2 header" box which contains several sub-boxes with different information regarding the image.

4. The "Code-stream" box with the image code-stream.

The first two boxes must appear in the order shown in Figure 2.10, and always at the beginning of the file. The other two boxes can appear in any place of the file, but always in the same order.

The "Code-stream" box also contains a set of sub-boxes in order to organize the different parts of the compressed content of the image. Within the set of these boxes, there are also some mandatory ones. The "Image header" box is mandatory and it must be always the first one. The boxes "Bits per component" and "Color specification" are mandatory as well, but they can appear in any place and order.

Part 2 defines new boxes to be included within a JP2 file, and a new file format, with the ".JPX" extension, also based on the JP2 structure. This new file format allows multiple codestreams and the definition of a composition of them. Furthermore, it allows for the inclusion of hyperlinks to external image files, so a JPX file might define a certain composition of several codestreams, one of them located in external files. This feature is used, for example, by the JHelioviewer project in order to generate the image sequences of the Sun for a specific time range.

CHAPTER 3

The JPEG 2000 Interactive Protocol

3.1 Introduction

Part 9 [45] of the JPEG 2000 standard is almost completely dedicated to the development of systems for remote browsing of images. It defines a set of technologies (protocols, file formats, architectures, etc.) that makes it possible to efficiently exploit all the characteristics of the JPEG 2000 compression system in the development of this kind of system.

The technology which Part 9 mainly focuses on is the JPIP (JPeg2000 Interactive Protocol) [80]. This protocol is the most used system at the moment of writing this document for the transmission of JPEG 2000 images. It was adopted by the standard as an evolution of the JPIK protocol developed by D. Taubman [78]. In this chapter the JPeg2000 Interactive Protocol is described and analyzed.

3.2 Architecture

The client/server architecture of a system for remote browsing of images that uses the JPIP follows the model/organization shown in Figure 3.1. As can be observed in this figure, the client is composed of four functional modules.

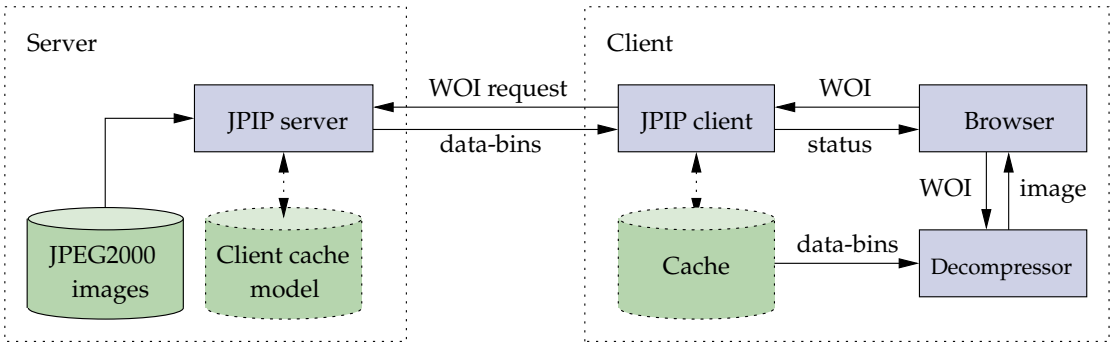

Figure 3.1: Client/server architecture proposed by the JPIP.

The Browser module consists of the interface which the user interacts with in order to define the desired WOI. JPIP allows the use of multiple parameters to define the WOI of the remote image. Most of these parameters refer to restrictions on the data partitions (number of quality layers, components, etc.). Geometrically, JPIP always requires that WOIs be defined as rectangular regions over a certain resolution level.

Once the user has defined the desired WOI, a request is sent to the "JPIP client" and "Decompressor" modules. The former is responsible for communicating and sending the WOI data to the JPIP server, using the messages and syntax defined by JPIP. When the server receives this information, it extracts from the associated image the required data for the reconstruction of the WOI.In JPIP, the necessary elements of the data partitions of a JPEG 2000 image are reordered and encapsulated in *data-bins*, which are the transmission data units. Data-bins associated with the requested WOI are sent from the server to the client.

As the client receives the data-bins from the server, it stores them in an internal cache ("Cache" module). The client cache is organized into data-bins as well. This cache is continuously read by the "Decompressor" module for generating progressive reconstructions of the WOI, which are passed to the "Browser" module to be shown to the user.

It can be seen that except for the Cache module which serves as a container, the rest of the modules, which compose the client, work in parallel. The Browser is continuously indicating to the JPIP client which WOI is requested by the user. The JPIP client keeps communicating to the server for transmitting this WOI and stores the received data in the cache module. The decompressor continuously generates reconstructions of the current WOI with the existing data in the cache. The modules begin their execution when the user defines the first WOI, and stop when all the data of the current WOI have been received. At any time, the user can ask for a new WOI without waiting for the completion of the previous one.

The client cache stores all the data-bins received from the server, for all the WOIs of all the images explored by the user. The server may optionally maintain a model of the content of this cache; this enables the server not to send data that the client already has. For instance, let us assume a user, after exploring a certain WOI_A, requests a new WOI_B that has some overlapping with WOI_A. If a server maintains the cache model of the content of the client cache updated, when the WOI_B is requested, the server only sends the data-bins necessary to reconstruct the part of WOI_B that is not overlapped with WOI_A. If all of the data-bins that permit the construction a certain WOI_i are defined by $D(WOI_i)$, we can say that the server only sends data-bins $D(WOI_B) - (D(WOI_A) \bigcap D(WOI_B))$.

The architecture proposed by JPIP has a special characteristic that makes it quite different from all other client/server architectures regarding the working philosophy. In the JPIP model, the server is responsible for the The server can modify almost any parameter indicated by the client without any limitations. The client has to adapt itself to the modifications made by the server for the requests sent. Therefore, the server can modify any WOI signaled by the client. Moreover, the server does not ensure that two identical requests will obtain the same responses. Although the content is the same, its organization and order can vary according to the server decision. This special characteristic of JPIP was designed to maximize the efficiency of the communication, taking into account that the server has direct access to the images and their contents. However, as will be explained later, this characteristic implies significant disadvantages.

The JPIP was designed to be independent of the used base protocol. However, the HTTP [27] is commonly used as base protocol, because of the similarity of the syntax with JPIP. When the HTTP is used there is a possibility of employing a secondary TCP channel for the data (HTTP-TCP mode). In this way, the requests as well as the responses are transmitted through the HTTP channel, but the responses do not include data, which are transmitted through the TCP channel. This working methodology is very interesting since it allows a faster and more fluent communication for the data of the image. Nevertheless, this working methodology is not widely used because, among other drawbacks, this communication method cannot be carried out when proxies are necessary [59].

3.3 Data-bin partition

Data-bins encapsulate different items of the partition of a JPEG 2000 image. This new partition of data defined by JPIP makes it possible to identify and gather the different parts of an image for their transmission.

When a client/server communication is started, the kind of data-bin flow that needs to be used must be defined. JPIP defines two kinds of flow: JPP, oriented to precincts, and JPT, oriented to tiles. The flow type indicates what kind of data-bins are used for the data transmission. The flow type most commonly used is the JPP, because the JPT flow causes the tiling effect shown in Figure 2.6.

Each data-bin is unequivocally identified with the image or code-stream it belongs to, the data-bin type and its identifier, and this information is included in the header of each data-bin.

Data-bins can be segmented for their transmission. A certain set of data-bins can be divided into a random number of segments, and they can be sent following any order.

Data-bin	*Flow*	**Contained information**
Precinct	JPP	All the packets of a precinct.
Precinct extended	JPP	The same content with additional content.
Tile	JPT	All the packets and markers of a tile.
Tile extended	JPT	The same content with additional content.
Tile header	JPT	The markers of the header of tile.
Header	JPP and JPT	The markers of the header of a code-stream.
Meta-data	JPP and JPT	Meta-data of an image.

Table 3.1: List of the data-bins defined by JPIP.

Each segment of data-bin includes the necessary information for its correct identification.

Table 3.1 describes the set of different types of data-bins defined in JPIP, showing which flow type they use and what type of information they save. Next, the most important types of data-bins are explained.

- **Precinct data-bin:** Contains all the packets of a precinct for a resolution r, a component c and a tile t. The packets within a databin are ordered by quality layer in increasing order.

  Every precinct data-bin is identified by an index I that is obtained by

  $$I = t + ((c + (s \cdot N_c)) \cdot N_t)\,, \tag{3.1}$$

  where N_c and N_t are the number of components and the number of tiles of the image to which the precinct belongs, respectively. The index of tile, t, as well as the index of component c and the value s begin with zero. A unique sequence number s is assigned to every precinct of a tile-component. This number starts with zero for the precinct of the lowest resolution level, located at the top-left border. It continues increasing by column and row, and covers all other resolution levels in order.

- **Tile data-bin:** Contains all the packets and markers associated with a tile. It is composed by concatenating all the tile-parts associated with a tile, including the markers SOT, SOD and all the remaining relevant markers. This kind of data-bin are identified by an incremental index, beginning with zero for the tile located at the top-left border.

- **Tile header data-bin:** Contains all the markers associated with a certain tile. It is formed by all the markers of the headers of all the tile-parts of a tile, except markers SOT and SOD. These data-bins are identified in the same way as the tiles.
- **Header data-bin:** Contains the main header of the code-stream, from the SOC (included) to the first SOT marker (not included). SOD and EOC markers are excluded from the header data-bin.
- **Meta-data bins:** These data-bins appear only if the associated image is a file of the JP2 family (.JP2, .JPX, .JPM, etc.). The meta data-bins contain a set of boxes of the image file (see Section 2.6). The standard does not define how these data-bins must be identified nor within which boxes they must be stored. When a meta data-bin is identified by zero, this means that it will include all the boxes contained in an image.

3.4 Sessions and channels

A client request to the server can be either stateful or stateless. Stateful requests are carried out within the context of a communication session, whose state is maintained by the server. Stateless requests do not require any session. The use of sessions improves the performance of the server because, for example, when establishing a session with a certain image file, the server opens that file and prepares it in order to be transmitted in data-bins; so all the requests associated with the same session do not cause the server to repeat the process of opening and preparing the image file. The stateless requests can be considered as unique sessions that finish when the server response ends.

Under a session, using stateful requests, the client can open multiple channels, making it possible to perform multiple stateful requests associated with the same session simultaneously. This is specially useful for applications that show simultaneously different regions of interest of the same image. The channels can be closed and opened independently, without affecting the session. Closing a session would involve closing all of the open channels associated with it.

Each session has a set of images associated with it. A session implies that the server maintains a model of the client cache. That model will only be maintained while the session remains active. A channel, for a certain session, is associated with a certain image and a kind of specific data-bin flow (JPP or JPT). The channel is identified in an unequivocal way by means of an alphanumeric string assigned by the server, and its format is completely free. Sessions are not identified, since the identifier of the channel must be enough to identify the channel as well as the session to which it belongs.

Many times, clients are interested in maintaining the client cache model in the server among several different sessions. The server by default ignores this possibility, starting a new session with a client, always with an empty cache model, although the client cache contains initial information. In this case JPIP defines a set of messages that allows the client to modify the model that the server has of its cache. Therefore, when the client starts a new session with a server, it would send a summary of the current content of its cache to improve the communication and avoid redundancy.

The caches of the clients generally have a maximum size limit, so they implement a kind of policy to remove those data-bins that are less used, like LRU (Last Recently Used). With the aim of maintaining the coherence of the cache model that the server has, the client must communicate to the server any change in its cache related to this policy.

The cache models maintained by the servers not only avoid redundancy between messages, they are also used in order to allow the client to perform incremental requests and control the flow of data. For instance, clients can indicate in their requests a parameter with the maximum length of the server response. Thanks to the cache model of the server, the client can perform successive identical requests, it but varies the parameters, so the server responds by only sending increments of information. Therefore the client has certain control and can adapt the flow of information to the available bandwidth and delay, but always taking into account that the server can modify the corresponding parameter at any time. This method of communication is indeed the most common one in the existing implementations of JPIP.

3.5 Messages. Examples

JPIP requests are composed of an ASCII sequence of pairs "parameter = value". This allows that a JPIP request can be encapsulated within a GET message of the HTTP [27], next to the character '?', concatenating all the pairs with the symbol '&'. Therefore, a typical HTTP request is produced/created when dynamic objects are referenced like CGI (Common Gateway Interface).

Some of the available parameters of a JPIP request are the following:

- **"fsiz=R_x,R_y":** Specifies the resolution associated with the required region of interest. The server chooses the smallest resolution of the image, so its dimension $R'_x \times R'_y$ satisfies that $R'_x \geq R_x$ and $R'_y \geq R_y$. Commonly, this parameter defines the resolution of the user screen.

- **"roff=P_x,P_y":** Specifies the position of the upper left border of the required region

of interest within the indicated resolution. If this position is not indicated, the server assumes the value $(0, 0)$.

- **"rsiz=S_x,S_y":** Specifies the size of the required region of interest. The server cuts this size in order to fit it into the real image according to the specified resolution.
- **"comps=C_0, C_1, ..., C_N":** Indicates the components of the region of interest the server must send. The rest of the components are ignored. If this parameter is ignored, the server sends all the existing components.
- **"len=number of bytes":** With this parameter, the client specifies to the server the maximum number of bytes it can include in the response. The server takes into account this limit, not only in the response to the current request, but in the rest of the next responses to the client within the same session.
- **"layers=L":** With this parameter the client can limit the number of quality layers the server has to send in its response.
- **"target=image":** This parameter identifies the image file from which to extract the specified region of interest. When the HTTP is used, this parameter is not necessary since the name of the image is obtained from the URL (Universal Resource Locator) specified in the GET message.
- **"cnew=protocol":** When the client wants to open a new channel under the same session, it uses this parameter, indicating the protocol that must be used for this new channel. The types of valid base protocols are "http" and "http-tcp".
- **"cid=channel identifier":** When the client creates a new channel, the server sends the channel identifier, which must be included in all the requests associated with that channel.
- **"cclose=channel identifier":** The client may decide to close certain a channel by means of this parameter, by simply specifying the identifier of that channel.
- **"type=flow type":** When a new channel is created, the client indicates what kind of data-bin flow is requested. The most important types of data-bin flows are JPP ("jpp-stream") and JPT ("jpt-stream").
- **"model=...":** As has been previously commented, the client may need to tell the server the contents of its cache in order to update the contents of the cache model maintained by the server. For instance, if

```
model=Hm,H*,M2,P0:20,-P1001
```

the client is telling the server that the cache model has to contain the main header of the code-stream, the headers of all the tiles, the meta data-bin number 2, which includes the first 20 bytes of precinct 0, and that precinct number 1001 must be removed.

The server JPIP responses are similar to the responses defined in the HTTP: first a ASCII part that contains a set of headers, and next, the data of the response composed by data-bins.

Some of the headers that the server can send to the client are the following:

- **"JPIP-cnew: cid=..."**: When the client requests to open a new channel, the server sends the associated identifier, as well as a set of data related to it. The server might include a host and port number which should be used by the client to communicate.
- **"JPIP-fsiz: R'_x, R'_y"**: The server defines the resolution size of the current window of interest. It may be different from that initially defined by the client. In fact, the server could modify any of the parameters defined by the client. For each "param" parameter specified by a client in a request, the server can answer by modifying its values by simply adding a header "JPIP-param". The client must always take into account the new values given by the server.
- **"JPIP-tid: target-id:** The server can assign a certain identifier to each target or remote image. When a new channel is established, it must tell the client the target identifier associated with the channel.

Next, an example of a sequence of messages between a client and a server is shown. The goal is to retrieve a WOI using a JPIP session over HTTP. Initially, the client sends to the server a request similar to the following one:

```
GET
/image.jp2?cnew=http&fsiz=800,600&rsiz=512,478&len=2000&type=jpp-stream
HTTP/1.1↩
Host:  www.jpip-server.com↩
↩
```

In this example, for the image "image.jp2", the client is requesting a WOI with a size of 512×478 that is within the maximum resolution level with a size equal or smaller than 800×600. It is also specified that the base protocol is HTTP, and that the maximum length of the response should be 2000. The type of data-bin stream is JPP, which is based on precincts.

The server would create a new session for the client (if the client has not already opened any other), in conjunction with a new cache model initially empty. It would then start a new HTTP channel. This would generate a response similar to the following one:

```
HTTP/1.1 200 OK↩
JPIP-rsiz:  500,400↩
JPIP-cnew:  cid=JP1↩
Content-type:  image/jpp-stream↩
↩
...  Data ...
```

As can be seen, the server gives the channel identifier "JP1" and changes the size of the WOI to 500×400. The server already includes in this first response the first 2000 bytes with data-bins of the required WOI.

Over and over again, the client will send the same request to the server until the desired WOI is completed, varying the parameter "len" if necessary for controlling the data flow. Therefore, in this example, the rest of the requests of the client would be as follows:

```
GET
/image.jp2?cid=JP1&fsiz=800,600&rsiz=500,400&len=1000&type=jpp-stream
HTTP/1.1↩
Host:  www.jpip-server.com↩
↩
```

Notice that the client must repeat the same WOI request, updated with the modifications specified by the server, with the exception of the channel identifier, and modifying the parameter "len". The cache model allows the server to send increments of data for the same requests.

3.6 Performance analysis

The JPIP introduces interesting characteristics for developing applications for remote browsing of images. Maybe the most noticeable is the simplicity of the client side, which becomes a mere intermediary between the user and the server. Although the proposed architecture for the client is rather sophisticated (four functional modules working in parallel), its implementation is simple, and the main processing load relies only on the decompressor module. Communication between the client and the server is

quite simple. All communication control is performed by the server. Clients only have to transform the WOIs specified by the user (by means of the browser) into formal JPIP requests.

Currently, this simplification of the client side is not justified. The processing load required to increase the intelligence and control of the client for handling the communication would never be similar to the existing load related to the decompressing and visualizing the WOIs. Any computer and/or device capable of carrying out these processes could also include more complexity and communication control.

The goal of looking for an efficient communication does not justify the simplification of the client side either. Theoretically, a simple request per WOI would be good enough for a communication without redundancy and with the minimum overload. In practice, for most of the implementations, the JPIP clients perform successive requests in order to receive the content of a WOI in an incremental way. Therefore, although the client maintains its simplicity, the communication is not completely efficient neither exempt from overload.

The simplification of the client side proposed by the JPIP negatively affects different aspects of a system for remote browsing of images, lowering its performance. The main negative aspects analyzed in this document are the following ones:

1. **Server scalability:**

   Applications for remote browsing of images, such as any common client/server architecture, must be as scalable as possible. The number of simultaneous clients must be as large as possible without causing a bottleneck. Simplifying the JPIP clients involves that the JPIP server achieves considerable complexity.

   Maintaining a cache model for each connected client involves a proportional increment of the memory consumed by the server. These cache models must index and store a mirror of the content, data-bin per data-bin, of the client caches.

   The more complexity the server implements, with the aim of achieving the maximum communication efficiency, the less scalable it is. For example, in some implementations the server modifies the requests of the clients and reorganizes the data to be sent, according to the available resources and bandwidth. This complex processing is less scalable.

   In this thesis two different approaches to this limitation are analyzed: i) Chapter 4 explains a technology for developing efficient remote browsing systems without requiring a JPIP server, but only a common Web server; and ii) the ESA JPIP server presented in Chapter 7, offers a highly-scalable open-source solution for a JPIP server.

2. **Prefetching support:**

   Nowadays the remote browsing systems incorporate more and more intelligence. They can learn from user behavior in order to offer a smooth and adapted navigation through the images. The interactions between the user and the images lose their atomic characteristic, that is, they are considered as a part of a defined path, making it possible to predict the next user movements and to stay ahead of the most probable requests. The server can send information to the client that requested it as well as other information that might be useful in the future, even though it has not been requested. This mechanism of prediction is called prefetching.

   In this context, A. Descampe et al. [21] have proposed a system where the information sent by the server is reordered taking into account not only its importance regarding the current requested WOI, but also its relevance for the next predicted WOI. This estimate is performed considering the previous requests of the user. Chengjiang Lin and Yuan F. Zheng [54] also proposed an approach to predict and anticipate user behavior depending on the requests previously performed.

   The proposal of Hao Liu et al [55] provides a semiautomatic navigation through high resolution images in mobile devices. It is a very attractive idea for these kinds of devices where the maneuverability is quite limited. Although the authors do not explicitly refer to prefetching, their work could easily be significantly improved by means of this technique.

   The prefetching can be implemented at the client side as well as at the server side. If it is implemented at the server side of the system, it becomes another factor for limiting the scalability of the server. Furthermore, the server should perfectly know the features of the user interface used in each client side in order to carry out correct predictions.

   The simplicity of the client makes it difficult to implement prefetching mechanisms on this side. Chapter 6 proposes an efficient technique for prefetching, fully JPIP compatible, for the special case of remote image sequences.

3. **Transmission efficiency:**

   As commented in Section 2.5, the LRCP is the most frequently used progression for ordering the packets when they are going to be transmitted, because it has been experimentally shown that it offers the best results compared to other progressions. In JPIP, the most frequently-used type of data-bin stream is JPP, which is based on precincts. A precinct data-bin stores all the packets of a precinct or-

dered by quality layer. This means that if a JPIP server applies the LRCP progression for the transmission, using the JPIP stream, it has to divide all the data-bins into as many segments as packets, reordering them according to the progression as well. This involves an additional data transmission overhead due to the addition of the data-bin header for each segment. Depending on the granularity of the JPEG 2000 image (code-block and precinct size, number of quality layers, etc.) and its size, this overhead may produce an overhead in the communication.

In order to reduce this overhead as much as possible, all the consecutive void packets are grouped in the same data-bin. This strategy is used in the ESA JPIP server implementation (see Chapter 7).

The approaches proposed in Chapters 4 and 5 can easily reduce, or even fully eliminate this transmission efficiency penalty.

4. **Coherence of the cache model:**

   The efficiency of the JPIP communication is directly related to the coherence of the cache model that the server maintain. The more exact this model is, with respect to the real content of the client cache, the better and the less redundant the communication is. There are situations where maintaining a good coherence of the cache model is quite difficult and could require an additional data overhead.

   When the client side runs on devices with memory restrictions, as for instance in mobile devices, it is necessary to release those data-bins less used from the memory. In this context, if maintaining a good coherence of the cache model is desired, the client should update it frequently by means of the previously-commented messages, provoking a data overhead in the transmission.

   It is almost impossible to maintain a good coherence of the cache model if: (i) the JPIP is used in error prone communication channels; or (ii) the loss of packets is rather probable, as for example in the case of wireless transmissions.

   For certain applications it is assumed that some errors or lost packets can occur during the communication and that the retransmission of these lost packets is not allowed. However, in the context of remote browsing applications over this kind of channel, the retransmission of data is not a drawback. The problem is that, under the philosophy of JPIP, a client cannot easily find out whether a lost packet has occurred or not, because it does not request explicitly each packet.

   The coherence problem does not affect when the client is who requests explicitly all the necessary contents to the server, as in the case of the approaches proposed in Chapters 4 and 5.

5. **Advanced user interfaces:**

   The user interface is a very important component of a system for remote browsing because it defines the way the user explores the images. There are different kinds of user interfaces for these systems. A wide revision of the possible user interfaces was done by Rosenbaum y Shumann in [68].

   The structure of the most frequently used interface integrates the functionality Zoom & Planning with the Detail & Overview (ZP-DO). This interface allows the user to explore the remote images by changing the zoom level, and within the zoom level, rectilinear movements. The Detail & Overview functionality is implemented by means of a reduced view. Therefore, the user can observe at any moment the complete image at a low resolution level, with the current WOI marked.

   With the interfaces ZP-DO, the user can define only rectangular WOIs, for a certain zoom or resolution level. Any other kind of interface that offers the user a higher degree of freedom, or that generates more complex visualizations, is not directly compatible with JPIP. For instance, the interface proposed by Ronsenbaum and Taubman [69] for improving the user experience with mobile devices would imply serious difficulties in order to be integrated into JPIP.

   In principle, the approach detailed in Chapter 4 might support any kind of user interface.

6. **Support of the Web caching proxies:**

   As has been previously mentioned, JPIP is commonly used over the HTTP, making it easy to establish communications through Web proxies. An instance of this is Squid [10], the most popular open-source implementation of a Web proxy server. In these cases, the JPIP specification always recommends including the appropriate headers and parameters defined in the HTTP to avoid the caching of the data in proxies. The reason is that the server responses cannot be associated unequivocally to the client requests, what would cause incorrect caching. In any case, the JPIP requests are formed as CGI URLs, and these kinds of URLs are usually discarded to be cached by almost all the proxies.

   The Web caching system is globally used on the Internet for improving the efficiency of the communications, removing the redundant requests that arrive at the servers, which are answered by the intermediate proxies. There is a considerable number of proxies, most of which are free access, and many of them transparent. For instance, most of the Internet providers use transparent proxies in order to reduce the volume of Web traffic of their clients.

In the case of systems for remote browsing of high resolution images, the data traffic between clients and servers is very redundant. This means there is a high probability that several clients perform requests of the same WOI, and/or several overlapped WOIs. The JPIP does not allow this redundancy to be removed using the existing Web proxies.

Although there are some works related to the implementation of a JPIP proxy [53, 60], none of them has been as widely used/tested as a common Web proxy server as, for example, Squid [10], which is widely present on the Internet.

Chapter 5 describes and analyzes an approach for a new protocol, based on the JPIP, which permits the efficient exploitation of the existing Web proxies, thus improving the final user experience.

CHAPTER 4

Remote browsing with the HTTP/1.1

4.1 Introduction

This chapter is devoted to describing one of the main contributions of this thesis, which consists of developing an architecture for interactive browsing of JPEG 2000 images based on the HTTP (HyperText Transfer Protocol), version 1.1. The main feature of this approach is that it completely dispenses with a specific server. It only requires an easy implementation based on the use of the HTTP/1.1 [27], widely used and tested on the Internet. The main advantage of this proposal is that the communication is managed entirely by the clients.

As previously commented, the JPIP usually works over the HTTP. The specification 1.1 of the HTTP is currently the most widely spread protocol on the Internet. Almost all the existing Web servers support this specification, although they state that they only support the previous 1.0 version because of compatibility issues.

The HTTP/1.1 has advanced characteristics like, for instance, pipe-lining, caching system or byte-ranging. The byte-ranging feature is specially interesting for remote browsing because it permits the extraction of a random range of data from a remote object. In other words, it allows remote random access. This means that within the same request, a client can ask for any set of ranges belonging to a certain file stored in a remote server.

A system for remote browsing of JPEG 2000 images can directly work with the HTTP/1.1, without requiring the JPIP. Clients can retrieve from the remote images all the ranges they need for reconstructing a certain WOI by means of byte-ranging. The main issue to tackle is how to efficiently determine which are these ranges. If the protocol used is JPIP, the server is in charge of determining this because it has the images locally accessible. Using a common HTTP/1.1 server, the client itself must perform this estimation remotely. Moreover, this should be done by downloading the minimum amount of data possible to avoid transmission overheads.

The next section is shows that some approaches regarding remote browsing of JPEG

2000 images using exclusively the HTTP/1.1 already exist. However, the approach proposed in this chapter (see Section 4.3) achieves better performance without data redundancy.

4.2 Related work

4.2.1 The proposal of S. Deshpande and W. Zeng

S. Deshpande and W. Zeng [23] worked on a detailed solution for the transmission of JPEG 2000 images just using the HTTP/1.1. Their proposal consists of creating an external index file associated to each image. This index file would have to be fully downloaded by a client before being able to access the related remote image. Once the index file has been downloaded, the client would build the index of the different parts of the image in the memory so that, for a certain WOI defined by the user, it could determine which segments of the file are necessary. These segments would be retrieved by means of byte-ranging requests to the Web server.

The most important features of this approach are: i) any file can be indexed, independently of the JPEG 2000 format, from simple raw J2C files to complex JPX ones; ii) no complex implementations are required at the client side.

The main drawback of this approach is due to the fact that it is necessary to fully download the index file before being able to send any request to the server. This initial delay, before any image reconstruction can be shown to the user, is directly proportional to the size of the index file and the available bandwidth. This delay may disturb the user of a remote browsing system.

S. Deshpande and W. Zeng proposed several file formats to be used for the index file, from the simplest one, with a small length, to the most complex one, which would occupy a larger space. From the point of view of the client requiring parts of an image regarding a certain WOI, the more complex the file format is, the faster and the more efficient the process to be carried out by the client is.

The simplest index file format should contain simply: (i) the main header of an image, (ii) the headers of all the tile-parts and (iii) the lengths of all the tile-parts and packets. The latter (iii) would be mandatory only if it is not already included within the headers; for instance, by means of either PLT or TLM markers. The authors of this approach have estimated that the average length of an index file, for an image, would be approximately 1% of the length of the image. Therefore, for instance, for a JPEG 2000 image with a length of 20 megabytes, an index file of 200 kilobytes is generated. This means that for a connection link with a bandwidth of 100 kilobytes per second, the client should wait for about 2 seconds before starting to send specific requests to

Box (Index)	Description
Code-stream	This box can contain any of the following boxes.
Header	Contains an index for the main header of a code-stream.
Tile-part	Contains an index of all the tile-parts of a code-stream.
Tile header	Contains an index for all the tile headers of a code-stream.
Precinct packet	Contains an index of all the packets of a precinct.
Packet header	Contains an index of all the packet headers of a code-stream.

Table 4.1: Index boxes defined in the standard.

the server. This initial delay is unacceptable for remote browsing systems.

Clearly, another disadvantage of the S. Deshpande and W. Zeng proposal is that the index file is separated from the image file. Although this does not affect the performance of the approach, it makes the management of the images more difficult.

4.2.2 Indexed JP2/JPX files

Although Part 9 of the JPEG 2000 standard is specially focused on the JPIP, it also includes a formal alternative to the index file proposed by A. Deshpande and W. Zeng. This approach consists of the definition of a set of JP2 boxes (see Section 2.6) that makes it possible to include the index of the main parts of a JPEG 2000 image within a file. By means of these index boxes it is possible to find out which file segments are necessary for reconstructing any desired WOI.

These boxes were initially proposed to facilitate the implementation of JPIP servers so they would be able to quickly extract the required data from the image files, just by first reading these boxes. Nevertheless, in remote browsing systems, these boxes can also be exploited using other protocols besides JPIP, such as the HTTP/1.1. Clients could first extract the index boxes from a remote image file, and then decide which segments to request. Table 4.1 shows a selected set of the main boxes that can be included in a JP2 file for indexing.

The Code-stream index box contains the position and length of a code-stream within a JP2 file. Within this box several kinds of boxes can be found in order to index each part of the code-stream (tiles, precincts, etc.). As in the case of the data-bins (see Section 3.3), the packets are not independently indexed, but rather grouped into precincts and sorted by quality layer.

The most relevant drawbacks of this kind of indexing, for its use in remote browsing of JPEG 2000 images with the HTTP/1.1, are that: i) a noticeable overhead in the

transmission is produced; and ii) a considerable number of round-trips are required for retrieving the index boxes and the data.

These disadvantages are mainly due to the structure defined by the indexing boxes. In order to index a code-stream, stored in a remote JP2 file, a client must first locate the Code-stream index box within the file, since its position is not fixed at all. This would imply reading blocks of 8 bytes (the length of the header of any JP2 box, composed by an identifier and its length) until the required box is found, and with the associated communication traffic that would be generated. Once this box is located, the sub-boxes it contains must be retrieved and processed for building the image index.

In order to build the index of the precincts, by means of the Precinct packet index box, 8 bytes per packet have to be read, 4 for the offset and 4 for the length. On average, to be able to build the precinct index of a tile, it should be necessary to read $(17 + 8 \cdot n_{pc}) \cdot n_c$ bytes, where n_c is the number of components of the tile and n_{pc} is the number of packets of each component. This does not take into account the initial bytes required for locating and processing the main box. This data overhead is rather inefficient.

Another drawback of the approach by S. Deshpande and W. Zeng is that it would be necessary to implement an application for generating the index. Currently, a JPEG 2000 implementation supporting the index generation does not exist, as has been defined by these approaches.

4.3 The JPL file format

The JPEG 2000 standard defines complex formats for the image files, making it possible to include a wide variety of additional information, apart from their own image data. However, in practice, for applications of remote browsing of images, most of this information is either unnecessary or can be organized in an independent way, separated from the image file. In most cases, a simple raw image file J2C, with only one code-stream without additions, can contain any kind of image for remote browsing.

The JPEG 2000 standard defines the set of markers (see Section 2.4) for indexing a code-stream, making the use of any other index unnecessary. The only existing obstacle is the tremendous flexibility allowed by the standard for defining this index in terms of structure and organization. Limiting this flexibility, and imposing some restrictions, it is feasible to define a simple image format for being used efficiently in a remote browsing system over the HTTP/1.1. Therefore, the external index file proposed by S. Deshpande and W. Zeng would not be necessary, nor would the data redundancy related to the index boxes defined in the standard.

During the research period elaborating this thesis, a new approach for remote

browsing using a common HTTP/1.1 server has been tackled. It uses a reduced file format called JPL [36, 32] that contains a code-stream with a fixed organization:

- The main header contains only the markers SOC, SIZ, QCD and COD. It also contains a TLM marker.
- The TLM marker is the last marker of the main header.
- Each tile-part contains only the markers SOT, SOD and PLT within its header. It contains all the necessary PLT markers for referencing all the contained packets.
- The LRCP progression is always the used one.
- Packets must be grouped in tile-parts by resolution level for each quality layer. Thus, for n_r resolution levels and n_l quality layers, a total of $n_r \cdot n_l$ tile-parts are obtained
- Precincts must be used with small enough granularity.
- The number of components must not vary among the different partitions of the image.
- Only one tile must be used.

These restrictions in the image file format do not ban practically any kind of JPEG 2000 image. Although this proposal deals only with simple image files, it might be extended to more complex formats like JP2 or JPX.

The JPL format has the advantage of being composed of many parts with a predefined length and known position. The TLM marker has a length equal to $6 + 4n_r \cdot n_l$ bytes. For each tile-part referenced by the TLM marker two values are included, the index and the length. The standard allows the use of either 0 or 2 bytes for the first value, with 0 being the value used in JPL. For the second value, JPL uses 4 bytes. The SIZ marker has a length of $40 + 3 \cdot n_c$ bytes, where n_c is the number of components. The COD marker has a length of $14 + n_r$ bytes. The QCD marker has a maximum length of $5 + 2 \cdot (3 \cdot n_r + 1)$ bytes. The header of most JPL files, without considering the TLM marker, can be obtained by retrieving only the first 512 bytes from the server.

Employing a tile-part division by resolution levels and quality layers, the packet index can be constructed sequentially, depending on the requested WOI, reading only the necessary PLT markers.

The efficient use of the JPL format in remote browsing systems together with the HTTP/1.1 requires an architecture similar to that proposed by the JPIP , but a little

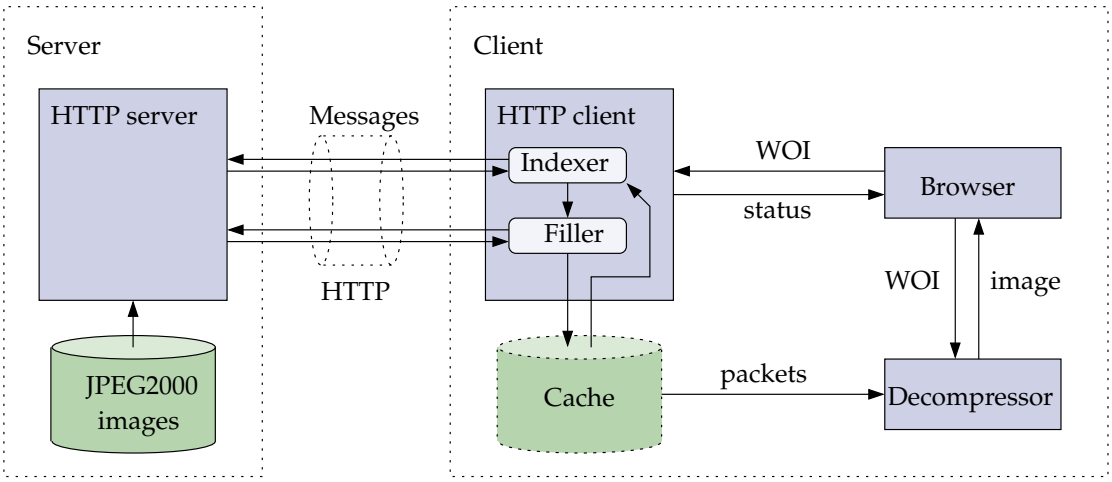

Figure 4.1: Architecture proposed for working with the JPL file format.

bit more complex on the client side. The proposed architecture client/server is as it appears in Figure 4.1.

The client module is composed by two different submodules which work in parallel, the "Indexer" and the "Filler" submodules. The index of the remote image is constructed in an incremental way according to the WOIs requested by the user. The Indexer submodule is in charge of constructing the index, starting with the main header, and reading the PLT markers as needed. When a user requests a WOI associated to a certain resolution level, this submodule determines if the corresponding PLT markers from the remote image have been read and processed.

The Indexer submodule is also in charge of estimating the necessary packets for reconstructing a WOI. The Filler is the submodule responsible for requesting from the server all the required packets using byte-ranging. The Filler retrieves the packets from the server and stores them within the cache. This cache feeds, apart from the Decompressor module, the Indexer submodule. Thus, the Indexer submodule can determine which packets have to be requested, and which ones are already stored within the cache. The general working scheme of these modules has been depicted in Figure 4.2.

Both submodules of the client use parallel communication channels with the server. This makes it possible to minimize the initial wait of the user as much as possible. Therefore, as soon as the Indexer submodule knows the exact location of a set of packets, the Filler submodule can start requesting them from the server.

The bandwidth is thus shared and it is possible to reduce the latency of the communication. Moreover, the Filler submodule does not have to wait for the response of a request sent to the server in order to send the next one, considering that the HTTP/1.1 protocol supports pipelining and persistent connections. Although this feature must be enabled in the server, most existing Web servers support it.

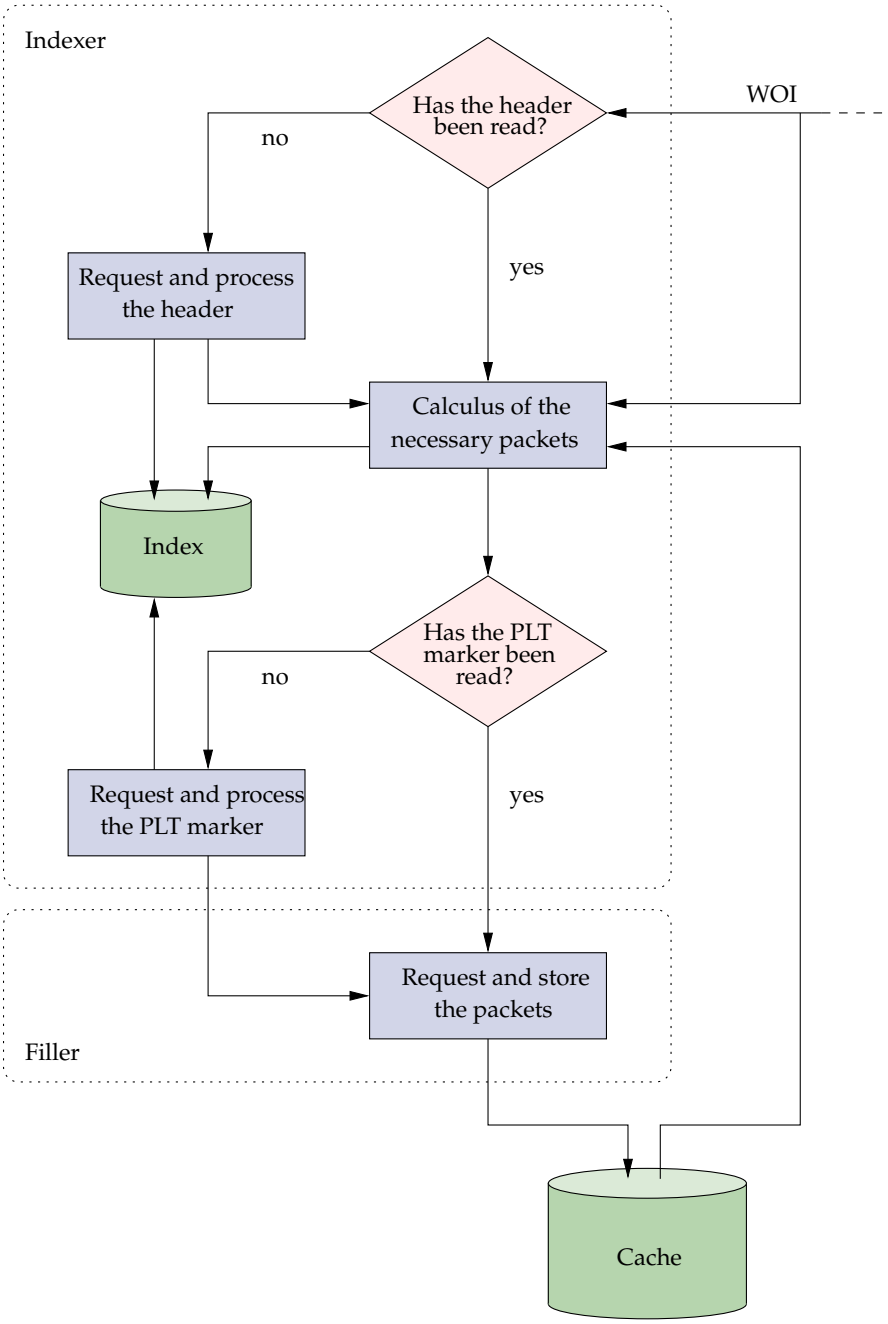

Figure 4.2: General working scheme of the client for JPL.

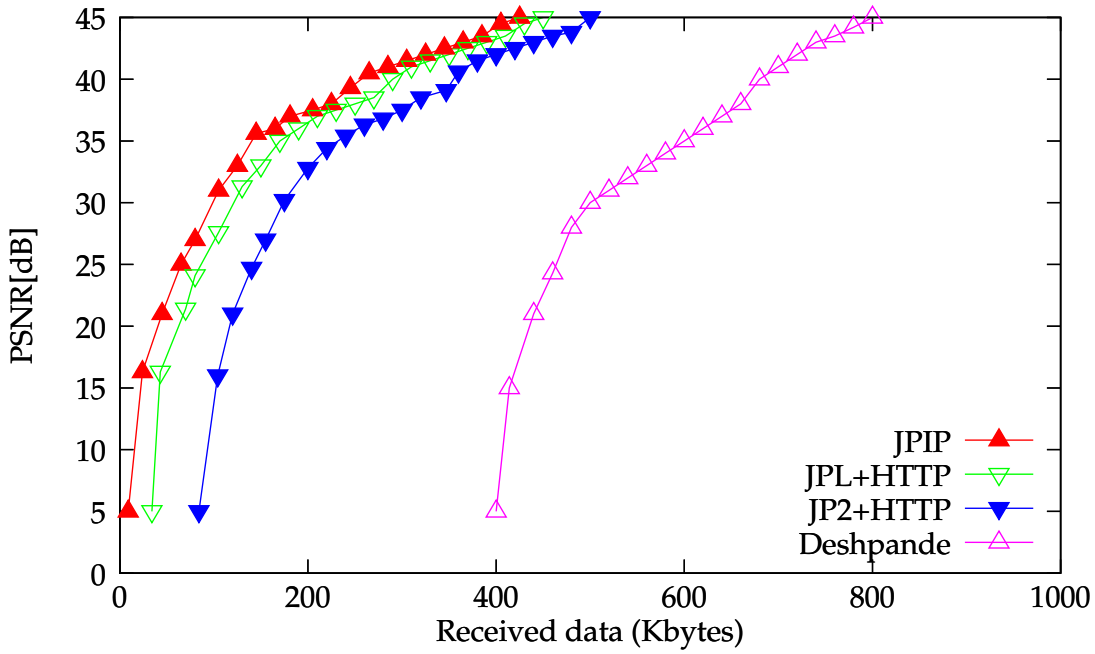

Figure 4.3: Comparison of the existing systems for remote browsing of JPEG 2000 images over the HTTP/1.1 with respect to JPIP.

A detailed description of the implemented classes and their connections to the previously-commented modules can be found within the repository of the Java source code [6], available at Launchpad.

4.4 Experimental results

This section shows the experimental results obtained from the evaluation of all the techniques discussed in this chapter. This evaluation has been done in terms of PSNR in dB (decibels) versus the amount of data received at the client side, which is the most commonly used metric for this kind of system. All the techniques described in this chapter have been evaluated and compared to the JPIP.

For this evaluation a wide set of different images has been used. The size of these images varies from 1024×1024 to 5462×7087. A precinct size of 128×128 has been selected, with a code-block of 64×64. A total of 8 resolution levels and 10 quality layers have been set. Lossy compression has been used. Experimental results shown in Figure 4.3 are the average values of the PSNR for all the images.

The JPIP implementation used for these tests is the one included within the Kakadu JPEG 2000 software suite [7]. A specific implementation for the proposal of S. Deshpande and W. Zeng was made because no other one exists. The approach based on the JPL file format, described in Section 4.3, has been implemented in Java [6]. For the case of the JP2+HTTP approach, it uses the same client structure as the JPL approach, but

only for building the indexes and reading and processing the associated boxes.

In Figure 4.3, experimental results of the evaluation of JPIP, JPL+HTTP, JP2+HTTP and Deshpande models are shown. As can be observed, JPIP offers better performance results than the other approaches. The JPL+HTTP option provides results that are, although slightly worse, hardly different from JPIP. The JPL+HTTP approach is a very attractive alternative to JPIP because of the following advantages: (i) no specific server is necessary and (ii) clients have control of the communication, with quite efficient performance. Moreover, JPL+HTTP overtakes the disadvantages of JPIP described in Section 3.6.

The rest of alternatives (JP2+HTTP and Deshpande) show a significant lower performance, specially in the case of the approach of S. Deshpande and W. Zeng.

CHAPTER 5

The JPIP-W protocol

5.1 Introduction

The approach of the format JPL on the HTTP/1.1 is very attractive because it solves many of JPIP's shortcomings and does not need to use a specific server. Nevertheless, JPL is not able to exploit efficiently the cache of the Web proxies because proxies do not manage the byte-ranging efficiently. For instance, as far as Squid is concerned, when some segments of a determined object are requested through byte-ranging, the whole object is always recovered by the server.

The performance of the applications that remotely visualize JPEG 2000 images can be considerably increased by making an appropriate use of the cache of Web proxies. The images used in this kind of system are rarely dynamic, such as those of Google Earth [4] or those described in [66]. A proxy could store these images, or parts of them, in its cache for a long time without updating them. During this period of time, the client requests should not reach the server because it could be solved by a proxy. Taking into account that the bandwidth among clients and proxies is usually bigger than that among proxies and the server, the response time can be reduced.

The response time can be decreased even more if redundancy is removed from the communication to the server, at the level of the overlapping among the WOIs requested by clients. That is to say, if a WOI of an image is demanded by a client, the proxy could serve data stored from other WOIs which have a certain overlapping and have been previously demanded by other clients.

This chapter describes the contributions of this thesis towards efficient use of the cache Web proxies in the field of remote browsing of images. In this sense, the research work has been based on the study of an effective communication protocol designed on the JPIP, called JPIP-W (JPIP-Web friendly) [34, 33, 35, 40].

5.2 The Web caching system

The Web turns the information into objects. Each object can have a different format and content (images, Java applets, etc.) which are identified by its MIME type (Multipurpose Internet Mail Extensions) [8]. Each object has a URL (Universal Resource Locator) as a reference. When a client asks the server for a particular object, he has to specify the corresponding URL.

A proxy Web is an entity which acts as an intermediary between a server and an HTTP client. The advantages of using these intermediaries for communicating are many; however, the most common one is that it provides a cache which facilitates the reduction of the client's response times and the traffic redundancy to the server.

When a client requires a specific object from a remote server, this request is received by proxies which determine whether the corresponding response is stored in its cache or not. If so, it is sent to the client without communicating anything to the server. Otherwise, it will transmit the request to the server. When the server response arrives at proxies, they will store it in their caches and retransmit it to clients. In this way, when another client demands the same object again, identified by its URL, the proxy will solve the request without making use of the server. Generally, the bandwidth between a proxy and its clients is much bigger than the one existing between the proxy and the server. This makes it possible to reduce considerably the response time if there are redundancies in the requests of the group of clients; a situation which is rather common.

The proxy has an "expiry date" for each object stored in its cache. Any client's request made before this date is solved by the proxy without using the server. The server is primarily responsible for determining the criteria of the expiration date of its objects. If the server does not specify it, the proxy will determine the most appropriate updating policy for the objects located in its cache. After this date, the object will not be sent to a client before being validated by the server.

Figure 5.1 shows the potential states in which an object can be found within the proxy cache and the conditions to produce a transition from one state to another. These potential states and their transitions are also described next:

1. An object is "Valid" when its contents are identical to the corresponding object stored in the original server. This means that the object can be served to a client without consulting the server at all.

2. After the expiry date, the object is automatically switched to "Invalid". In this condition the object must be validated by the server before being sent to any

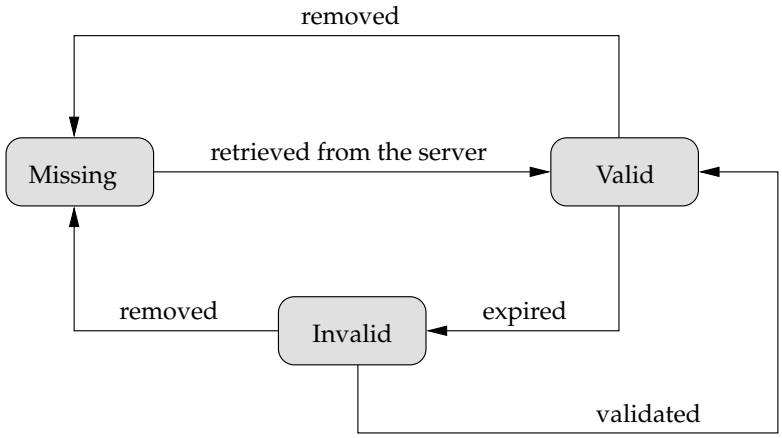

Figure 5.1: States and transitions of an object in a Web cache.

client. This validation consists of the consultation by the proxy to the server in order to know if the object has changed or not. If it has not changed, the object switches again to "Valid", thereby updating its expiry date. If it has changed, the proxy should obtain the object of the server again.

3. Due to a lack of space in the cache, the proxy can remove from its cache one or several objects according to some established policy, such as LRU (Last Recently Used).

The caching made at a proxy level can be controlled, to a certain extent, by the server. By default, all the objects served by the proxy are handled and stored in its cache by a predefined policy. The server can include headers in HTTP messages to tell the proxy how to administrate a determined object (expiry date, caching areas, etc.)

Amongst the most frequently used headers, the most important are those which can indicate the expiry date, or the time when a determined object is valid in the cache. There are two HTTP headers which allow this: `Expires` and `Cache-Control`.

The first one allows the server to clearly specify the expiry date of a determined object. An example of server response with this header would be:

```
HTTP/1.1 200 OK↩
Date:  Sat, 5 May 2007 10:23:24 GMT↩
Last-Modified:  Sat, 4 May 2007 08:00:35 GMT↩
Expires:  Sat, 5 May 2007 11:23:24 GMT↩
↩
<Object>
```

In this answer, which the server would send to the proxy, the `Expires` header would show the latter that the object should remain valid until 5/5/2007 at 11:23:24. Until this

date the proxy will not have to consult the server about the mentioned object again. The `Date` header is usually included in the answers in order to show the date of the server. This is useful, for example, in order to know the time that the object needs to cross the network. The `Last-Modified` header shows the date when the last modification was made in the object.

The main disadvantage of `Expires` header is that a particular synchronization is required between the server and the proxy clocks in order to accurately know when an object expires. Alternatively, the `Cache-Control` header offers the the possibility of showing an expiry date related to the sending time. Therefore, an example of the previous answer with this header would be:

```
HTTP/1.1 200 OK↩
Date:  Sat, 5 May 2007 10:23:24 GMT↩
Last-Modified:  Sat, 4 May 2007 08:00:35 GMT↩
Cache-Control:  max-age=3600↩
↩
<Object>
```

With this header the expiry date of the object is indicated in seconds, 3600 in the example, in relation to its reception.

When the object switches to "Invalid", the proxy can validate it again by making a request to the server, which includes the `If-Modified-Since` conditional header. This header will tell the server to return the object only if it has varied since the date shown. For example, if the proxy wanted to recover or revalidate the obtained object in the previous example after the expiry date, it would make a request similar to the following:

```
GET /object HTTP/1.1↩
Host:  server↩
If-Modified-Since:  Sat, 5 May 2007 11:23:24 GMT↩
↩
```

If the object has not changed since the date shown, it will send back a response with no data, similar to the following:

```
HTTP/1.1 304 Not Modified↩
Date:  Sat, 5 May 2007 11:24:24 GMT↩
Cache-Control:  max-age=3600↩
↩
```

In the response, the server provides a new expiry date for the object.

The `Cache-Control` header is also useful for specifying which degree of "cacheability" has a determined object. By indicating, for example, the `no-cache` value in the header, the server forbids the object to be stored in any cache. This is the header suggested by the standard JPEG 2000 for working with JPIP, thereby avoiding any possible interaction with the proxies. To explicitly declare that an object can be managed in any cache of any proxy, the same header is used but with the `public` value.

5.3 Proxy caching support

In order to support the cache system of the Web proxies efficiently, the protocol JPIP-W thoroughly changes its philosophy, despite being designed as a JPIP superior layer.

First of all, the partition of the image in data-bins is not considered because, as commented in Section 3.6, this data structure of a JPEG 2000 image is not appropriate when the LRCP progression for the transmission is used. However, this is the best progression for the transmission. Alternatively, through JPIP-W, the compressed content of an image is divided into minimum length segments called *blocks*. For each image, a minimum value S_B for the block size is defined (in bytes). Blocks generated by JPIP-W will have a size such that blocks include the minimum number of packets whose total size is higher than or equal to S_B. The S_B value is established by the client at the beginning of the communication, being modified by the server if necessary.

Every block of an image will be treated as a Web object, independent of the server, clearly identified through an incremental numerical index. This index would start from zero, which would contain the main header and would increase one by one sequentially for each image block. The address of the image, the index of the block of the same image and the S_B value used to generate this block will allow the formation of a URL which can be used by a client to recover the information of this block.

The divided blocks allow the client to independently access the different parts of a remote image, as would be done with the byte-ranging method of the HTTP/1.1. However, whereas the byte-ranging is not very suitable for being "cached" by the existing Web proxies, the blocks defined by JPIP-W are.

The reason for not using the elements of the code-stream directly and putting them together in blocks is to minimize the overload of information of the HTTP headers. If we could remotely reach, for instance, every packet of the image independently, by bearing in mind that the size of the same elements can vary substantially (they can also be one byte in size), the overcharge introduced through the HTTP headers would be significant. By defining a minimum size of blocks, we establish an average maximum percentage of overcharge caused by the headers; i.e.; if the HTTP headers added to

the client-server communication account for an average of 100 bytes per message, we would have a maximum overcharge of $(10000/S_B)\%$ per message.

Anyway, the use of blocks introduce some useless overhead. When a client obtains a specific block from the server to rebuild a WOI, it is possible that not all the elements of that block are necessary or useful for that WOI. The average overcharge of useless data for each block depends on the size of the image and of the S_B chosen value for the same image. This is why S_B is a value that strongly affects the JPIP-W efficiency.

The JPIP-W server has to guarantee that for a special S_B value the blocks of an image change neither their contents nor the organization. This means that two requests made during different periods of time, that have the same URL of blocks have to generate the same answer from the server. This is the way the blocks of an image can be managed correctly by the Web proxies. The server will include the appropriated HTTP headers we have mentioned in the previous section to guarantee the "cache- ability" of the blocks by showing the expiry time.

Through JPIP-W the interchange of messages that the client makes with the server to restore the information of a determined WOI is different from JPIP. There are two types of messages: the index ones and the block ones. The former is used by the client to show the server which WOI they want to visualize, which remote image S_B value he needs to use. This message has a syntax which is identical to those of JPIP requests. The server processes the image and sends back to the client the list of blocks which are necessary to rebuild the WOI, with the necessary information to be processed. While the client receives this list, he asks for the blocks (one by one) from the server. In this way, blocks of the same image can be shared at the level of the cache of the proxy, among requests from different clients.

To easily realize how different the JPIP and JPIP-W work, Figure 5.2 shows an example of the interactions and operations of JPIP and JPIP-W. Here, the lines of time between the interaction of the two clients (A and B) and the server are shown. According to the two protocols, client A recovers a WOI X ("GET WOI X" in the picture) and later client B recovers another WOI Y ("GET WOI Y" in the picture). The WOI Y shares the data-bins 0, 1 and 2 with X. In the communication with JPIP-W there is an intermediate proxy cache, but with JPIP there is not.

To simplify this example, it has been considered that all the datagrams have the same size. The bandwidth is constant in time and the bandwidth of the proxy and the client is much bigger than the bandwidth between the proxy and the server. The latency of the network is constant. The size of the data-bins has also been considered constant and those JPIP-W blocks contain exactly 5 data-bins, and one of them is useless for the WOI. As can be seen in Figure 5.2, for WOI X the interaction with JPIP is

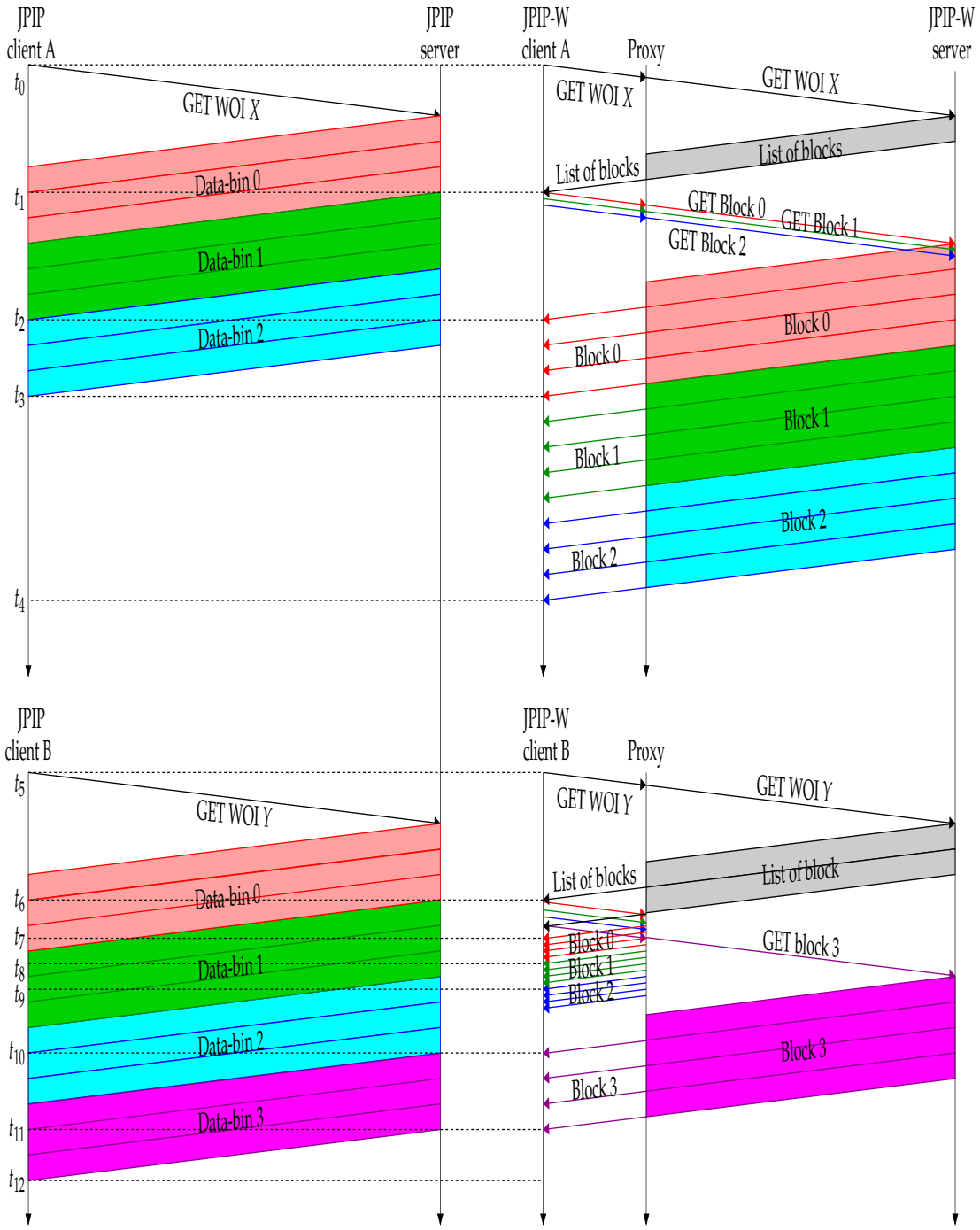

Figure 5.2: Example of the data interactions and operations of JPIP and JPIP-W.

faster than JPIP-W, basically because: (i) the cache of the proxy is at first empty, (ii) the list of the blocks has to be received before these can be obtained and (iii) the blocks usually include a percentage of useless data due to, for example, the protocol headers. Nevertheless, for WOI Y the transmission with JPIP-W is faster than that of JPIP due to the mediation of the proxy. For instance, at time t_8, client JPIP-W has recovered from datagram of block 1, while client JPIP is still receiving the first datagram of data-bin 1.

As far as the architecture is concerned, JPIP-W uses the same as JPIP, however, the communication and the administration of the information would be made through blocks and not through data-bins. To best reduce the initial latency produced by the list of blocks (Figure 5.2), an architecture is used for the client which is similar to that proposed for the JPL file format (Figure4.1). It means that the module of the client is divided into two sub-modules which use two different communication channels. The first sub-module would be in charge of the index requests while the second would use the block requests.

5.4 Messages. Examples

When a WOI has been defined by a user, the JPIP-W client will ask the server the entire index block using a typical JPIP but through a `JPIP-W` header and the `yes` value. The request will be similar to the following:

```
GET /image.j2c?rsiz=640,480&roff=0,0&fsiz=1024,768&bsiz=1000 HTTP/1.1↩
Host:  get.jpeg.org↩
Connection:  keep-alive↩
JPIP-W: yes↩
↩
```

Through this request the client will be asking for the entire index block which allows rebuilding the WOI situated in $(0, 0)$ with a size of 640×480, which is within the resolution level, less or equal to 1024×768. In order to divide the image into blocks, the server would use a value of 1000 bytes as the minimum value of the block size. This value is shown by the `bsiz` parameter, which is not defined in JPIP.

The client includes the HTTP `Connection` header with the `keep-alive` value to guarantee persistent connections. Although through HTTP/1.1 the connections are usually persistent, many proxies need the explicit use of this header to use this kind of connections The performance of the JPIP-W decreases significantly if persistent connections are not used.

When the JPIP-W server receives the client's request, it will divide the associated image into blocks (if it has not been done already) using the specified minimum block

size. When the image has been divided into blocks, the server estimates the set of blocks which are necessary to generate the requested WOI, and sends the block indexes as an answer. So, for example, the server will be able to answer a message similar to the following:

```
HTTP/1.1 200 OK↩
Connection:  keep-alive↩
JPIP-bsiz:  500↩
JPIP-W: yes↩
↩
<vbas(4)><vbas(58)><vbas(0)>
<vbas(6)><vbas(22)><vbas(83)><vbas(0)>
...
```

The JPIP-W server once again includes the `JPIP-W` header in the answer through the `yes` value. This allows the client to check if the server with whom it communicates really withstands the JPIP-W. The client's request would have been easily processed by a JPIP common server. This would have ignored the parameters and headers it does not know and would have generated a proper JPIP answer, but without including the `JPIP-W` header so that JPIP-W and JPIP can coexist without any problem. In the example, one can see how the server has varied the minimum block size established by the client, from 1000 to 500, through the `JPIP-bsiz` header. Only this kind of JPIP-W requests can be modified by the server.

Even though a more complex codification method for the block indexes could have been chosen, a simpler standard efficient method has been chosen instead. The block indexes codify themselves incrementally by using the VBAS format (see Section 2.4). For each block the necessary increments or offsets which allow to extract the important blocks information are included. These offsets are encoded as the block indexes. The offsets list is included after each block index; it has to contain at least an element and end in 0. In the previous example, we would have the block index 4 with the associated offset 58 and the block index 10 ($4 + 6$) with the offsets 22 and 105 ($22 + 83$).

The sent offsets are necessary to extract and process the packets which contain the blocks. As already described in Section 2.5, the JPEG 2000 packets are not completely auto-contained and can depend on other packets to be decoded. When the server sends the blocks indexes associated to a WOI, it has to include the corresponding offsets so that the client can codify the included packets, without having received other blocks. The offsets allow jumping the packets which do not belong to the WOI and that, as they cannot be decoded, prevent the contiguous packets from being decoded.

The pseudo-code that would allow the client reading the block indexes and their corresponding offsets would be as follows:

```
iblock = 0
while not end of message
  iblock = iblock + read_vbas()
  offset = read_vbas()
  next_offset = read_vbas()
  offsets[iblock].add(offset)
  while next_offset != 0
    offset = offset + next_offset
    offsets[iblock].add(offset)
    next_offset = read_vbas()
  end
end
```

With the previous code, a `offsets[iblock]` list with all the offsets specified by the server would be created for each `iblock` block index. The function `read_vbas` would return the following numerical value of the answer of the server, decoding VBAS.

The block indexes list that the server sends back never includes the zero index. This index corresponds to the block which contains the main header of the image and the client will be the one that will decide to request it, if it has not done so prior.

As soon as the client specifies the first block index of the list, it will ask for it from the server. For example, to request the block 4 a message similar to the following would be used:

```
GET /image0.j2c/bsiz/500/blk/4 HTTP/1.1↩
Host:  get.jpeg.org↩
Connection:  keep-alive↩
↩
```

The URL used to give a reference to the block has the form of a path file. Unlike the syntaxes used in the requests of JPIP, this form defined by JPIP-W prevents the rejection from the Web proxies to store the objects in the cache.

An example of an answer of the server for a block request could be:

```
HTTP/1.1 200 OK↩
Cache-Control:  public↩
Date:  Sat, 5 May 2007 10:23:24 GMT↩
```

```
Expires:  Mon, 7 May 2007 11:00:00 GMT↩
Connection:  keep-alive↩
Content-Length:  1340↩
Content-Type:  application/octet-stream↩
↩
...
```

It can be seen that the server includes the expiring time associated to the block through the `Expires` header. It uses the necessary `Cache-Control` and `Connection` headers to assure the "cache-ability" of the object and the persistence of the connection. It also includes the HTTP `Content-Length` and `Content-Type` headers to provide information on the length and the type MIME of the data respectively.

The pseudo-code that will allow codifying the blocks received from the server would be the following:

```
ipacket = initial_packet(WOI)
for each iblock
  ioffset = 0
  init read block[iblock]
  while not end of block[iblock]
    if (ioffset == 0) or
       not included_packet(WOI, ipacket)
      jump offsets[iblock][ioffset] bytes
      ioffset = ioffset + 1
      ipacket = next_packet(WOI, ipacket)
    end
    extract packet
    decode packet
    store packet in the cache
    ipacket = increase_packet(ipacket)
  end
end
```

The `initial_packet` function would recover the index of the first packet of the image associated to the requested WOI. The client can calculate this index by processing the main header of the code-stream. Actually, the client can value which packets are needed to visualize a specific WOI by only processing this header. The indexes of the packets would be assigned incrementally following the progression used to compress the image.

With JPIP-W it is possible to use any kind of progression to compress the image although it would be ideal to use the LRCP progression. The server could also use a progression, for instance RLCP, to store the image and a different one, like LRCP, for the transmission although this would require an on-line codifying process.

The `included_packet` function checks if the packet specified by its index is associated with the indicated WOI. The `next_packet` function will return the index of the following packet, which belongs to the WOI as well. Finally, the `increase_packet` function would return the index of the following packet to that specified as parameter.

5.5 Experimental results

To evaluate the JPIP-W we have designed a scenario where a server can communicate on the HTTP with two different clients: A and B. Between the server and the clients there is a proxy Web with cache. The bandwidth between the server and the proxy has been fixed at 100 kilobytes/s, which represents a typical ADSL connection, whereas the bandwidth between the proxy and the clients has been fixed at 100 megabytes/s, which corresponds to a LAN connection.

In this scenario, the JPIP-W has been compared to the JPIP in terms of the PSNR and the transmission time. The JPIP implementation provided by Kakadu JPEG 2000 [7] has been compared with our proposal.

In the server, eight natural test images of the standard ISO 12640-2 [42] have been stored and compressed with JPEG 2000 in a raw J2C format with the following parameters:

- Lossy compression
- Precinct with a size of 128×128.
- Code-block with a size of 64×64.
- 16 quality layers.
- 8 resolution levels.

The images of the standard are divided into two groups: group I is formed by three images with a size of 3072×4096 (“woman with glass”, “fishing goods” and “silver”) whereas group II is formed by five images with a size of 4096×3072 (“flowers”, “Japanese goods”, “field fire”, “pier” and “threads”). Although these test images have homogeneous resolution, their compressed size oscillates between 3 and 12 megabytes. All the results given in this section are the average values of the results obtained for these 8 test images

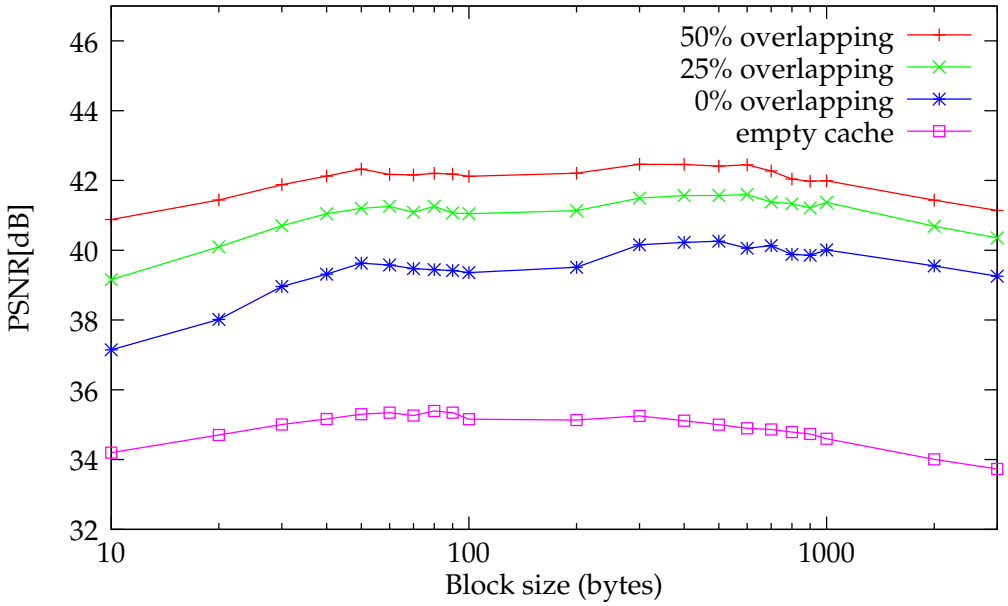

Figure 5.3: JPIP-W performance for different values of the block size and different values of the percentage of WOI overlapping. Average values of the PSNR over all the images are given.

The experiments have been divided into three different categories. In the first experiment the best minimum block size has been estimated in order to be used with the group of the test images, for the remaining experiments. To do that, the PSNR obtained has been recorded in client B for WOI $(0, 0, 700, 700, 0)$ after 3 seconds of transmission with the JPIP-W, considering four different situations: for one of these situations the cache of the proxy is empty, while for the other three the cache of the proxy contains the blocks associated with WOIs with an overlap, as far as the one corresponding to client B is concerned, of 0%, 25% and 50%. It is assumed that these WOIs have been previously requested by client A. Minimum block sizes from 0 bytes to 10 kilobytes have been evaluated. Figure 5.3 shows the values of the PSNR obtained from these experiments.

Experimental results show that for the four situations the maximum PSNR values are obtained for block sizes between 100 and 1000 bytes. Values of the block size smaller than 100 bytes produce a decrease of the PSNR due to the overload of the HTTP headers. Above 1000 bytes, the decrease of quality is due to the overload of useless data for the WOI. For the rest of the experimental results shown in this section, 500 bytes have been used as the value of the minimum block size.

In a second set of experiments, JPIP is compared to JPIP-W under three different overlapped scenarios. It has been assumed in all the experiments for both protocols (JPIP and JPIP-W) that client B has recovered the WOI $(0, 0, 700, 700, 0)$ from the server. For the experiments with JPIP-W, the same three situations of the WOIs overlapped

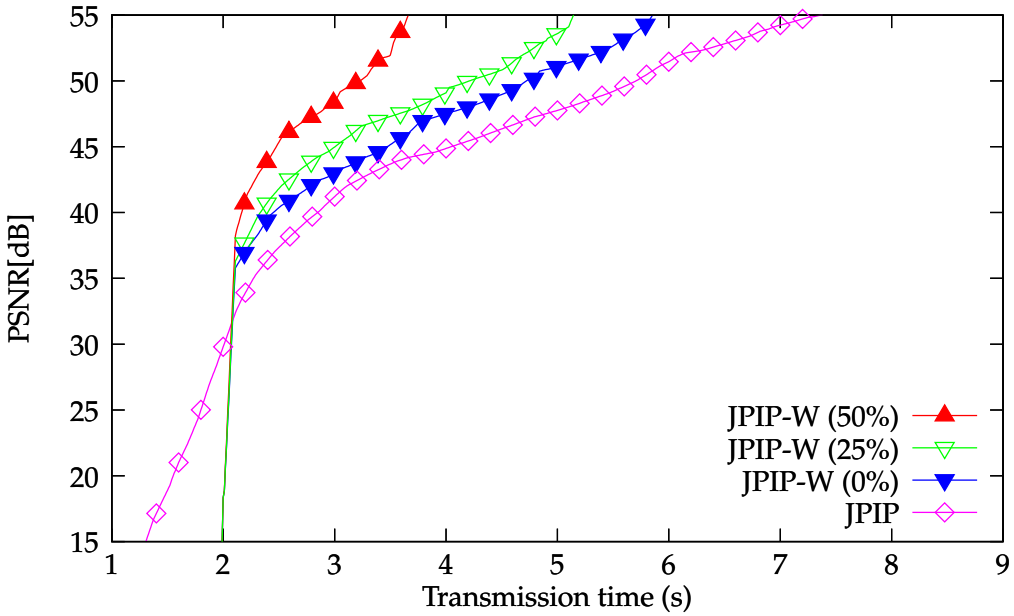

Figure 5.4: A comparison between the JPIP-W and the JPIP in terms of the PSNR versus the transmission time, and using three different degrees of overlapping. Average values of the PSNR over all the images are given.

and stored in the proxy, as described in the previous experiments, have been considered. Values of the PSNR versus transmission time have been drawn in Figure 5.4. It can be seen that JPIP-W outperforms JPIP as soon as there is a minimum amount of information in the cache of the proxy.

In this figure we can observe how JPIP-W is superior to JPIP at any overlapping level. Although JPIP permits to reconstruct the WOI sooner than JPIP-W, after two seconds from the beginning of the transmission, the quality reached by JPIP-W is considerably higher than that of JPIP. It should be highlighted that, even for a value of the overlap of the WOI stored in the proxy of 0%, JPIP-W is better than JPIP. This occurs because there are blocks which belong to lower resolutions that are shared by the WOIs.

In the last set of experiments, a remote visualization session made by a user situated in client B has been considered. We assume that previously, another user situated in client A, has made a remote visualization session. We have used a browser which works in the same way as the one which is included with Kakadu (something typical among the existing browsers). As soon as this browser has selected a remote image to explore, it recovers the maximum complete resolution of the same image whose dimension is equal or inferior to the screen resolution of the user. In the experiment we have considered a user screen resolution of 800×600. Although this resolution is not used very much nowadays, it makes it possible to homogeneously compare two groups of images by considering the different dimensions.

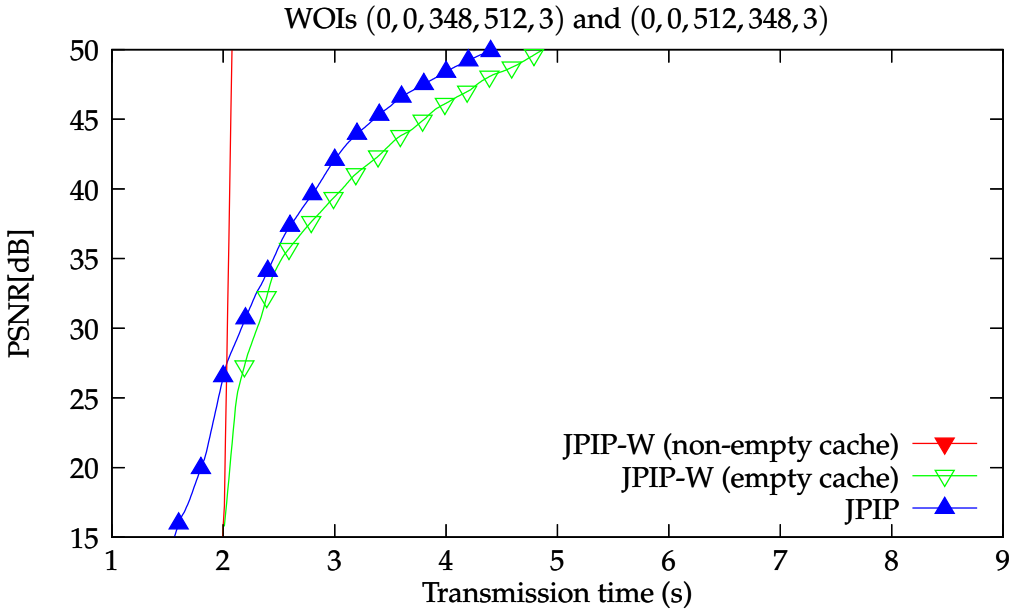

Figure 5.5: Comparison between the JPIP and the JPIP-W for the first browsing movement.

The session previously made by client A only consists of a first visualization, that is to say, it recovers the WOI $(0,0,348,512,3)$ for images of group I, and the WOI $(0,0,512,348,3)$ for images of group II. In the case of the JPIP-W, these WOIs will be stored in the cache of the proxy when client B starts his session.

By considering the session previously made by client A, we show the results of the three sequential movements made by client B during his session. During the first movement, client B would recover the same WOI as client A. The results appear in Figure 5.5.

In the following movement client B zooms in the inferior right corner, without modifying the dimension of the desired region. This means that it can recover the WOI $(348,512,348,512,2)$ for group I and the WOI $(512,384,512,348,2)$ for group II. Figure 5.6 shows the results.

In the last movement client B zooms in the inferior right corner again, without modifying the dimension of the desired region. The WOI $(1152,1536,348,512,1)$ is recovered for group I and the WOI $(1536,1152,512,384,1)$ for group II, as we can see in Figure 5.7.

During these three movements of client B's session the result of the same B session has also been included in the comparison but considering that the proxy does not have any information stored in its cache.

As can be seen, during all three of client-B's movements the performance of the JPIP-W is quite superior to that offered by the JPIP. The difference during the first

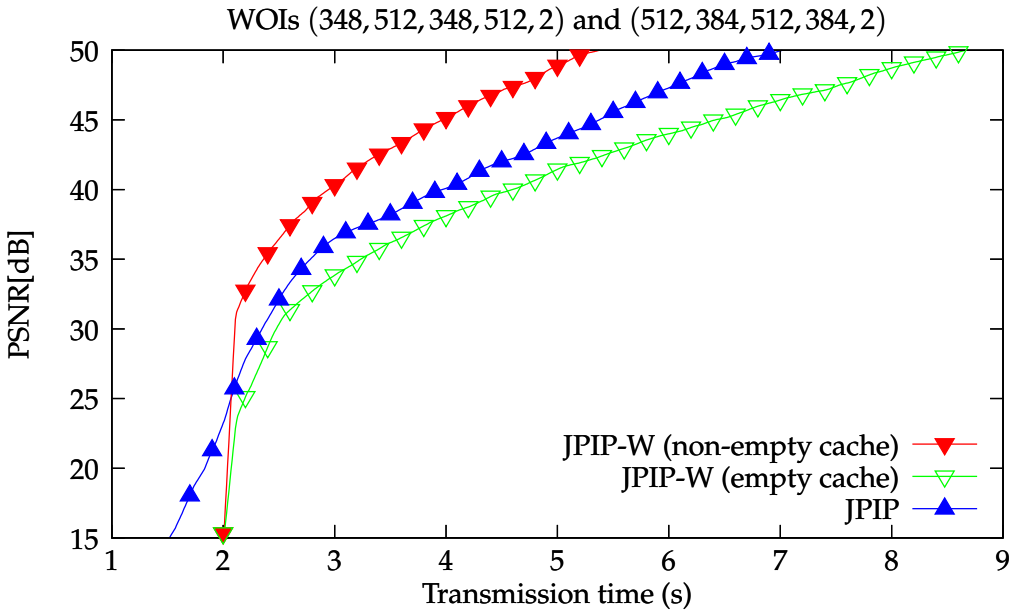

Figure 5.6: Comparison between he JPIP and the JPIP-W for the second browsing movement.

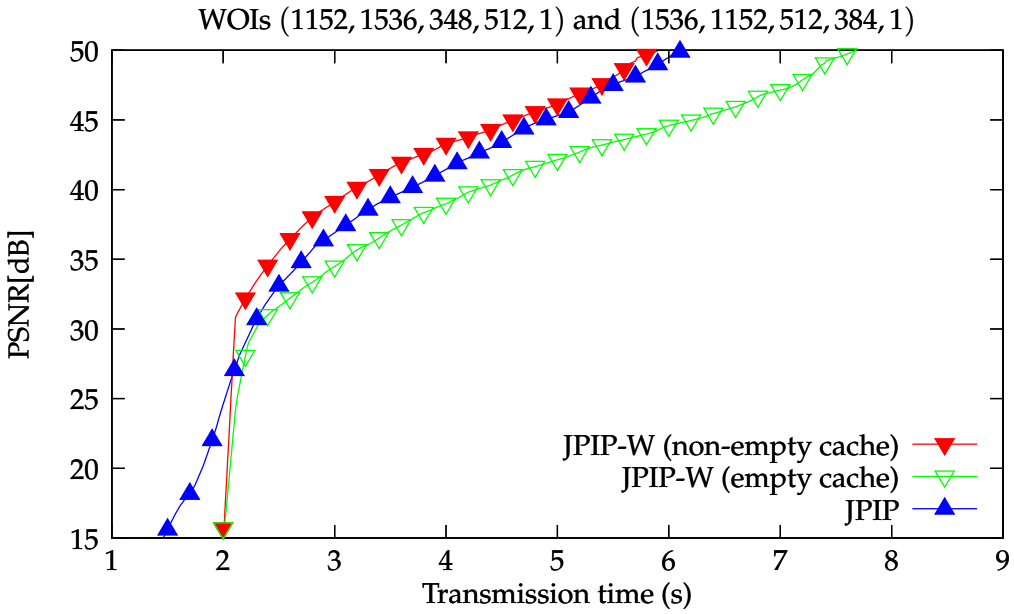

Figure 5.7: Comparison between the JPIP and the JPIP-W for the third browsing movement.

movement is enormous. This result during the first movement would be produced for all the images previously accessed by any other users, which is something typical in any remote visualization system.

At worst, that is to say, when there is not any information in the cache of the proxy, the JPIP-W performance is inferior to that of the JPIP, but quite near.

CHAPTER 6

Prefetching of image sequences

6.1 Introduction

An interesting field of application for JPEG 2000 is astronomy and, more specifically, the JHelioviewer project. The basic functionality of the JHelioviewer application (see Section 1.2) allows users to explore the available data at any given time. An interesting extension of this functionality is that it enables users to move smoothly through a sequence of time-coded solar images during a specific time range. This type of functionality could also prove to be very valuable in other domains such as tele-medicine and tele-microscopy.

However, viewing JPEG 2000 image sequences is both computationally and bandwidth intensive, which often compromises the quality of the viewing experience. This compromise manifests itself to the user as lack of responsiveness, i.e.; as a choppy image rendering. To address these challenges, we propose a special prefetching strategy that enables users to view image sequences with smooth transitions and without experiencing any penalties in responsiveness or quality gaps.

6.2 Related Work

The efficient access to large image files over networks has been an active research topic for a long time, in particular because images account for a considerable fraction of the total network traffic. Caching has been historically recognized as one of the most promising techniques to reduce bandwidth usage and server load and to improve performance. Several caching and prefetching schemes and algorithms have been proposed to reduce network traffic and minimize access delays [26, 17, 16]. More recently, image-specific caching techniques have been proposed to take advantage of the memory and processing capabilities of modern client systems and consequently to expedite image retrieval [75, 76, 82]. The JPEG 2000 standard has introduced new capabilities that can also be leveraged to improve some of the existing caching techniques.

Extensive work has been done in the area of accessing JPEG 2000 images over HTTP to enhance the user's interactive browsing experience [22]. Some approaches use a dynamic traffic regulating mechanism [52], while others employ a virtual media protocol to prioritize the compressed bit stream of the region of interest (ROI) [51].

Other approaches have focused on prefetching techniques using prediction-based server scheduling and cache management algorithms [54] or quad-tree-based indexing techniques, which take advantage of the space-frequency localization property of wavelet transforms and support subregion access [74, 67].

This notion of subregion access for streaming a WOI seems promising in enabling efficient, demand-driven browsing, which allows clients to quickly access regions of interest from voluminous images. Still, given the inherent client/server exchange latency, the browsing experience of JPEG 2000 images can be further improved by early fetching of future WOI data.

Some approaches use heuristic mechanisms that improve browsing responsiveness [54], using a formal rate-distortion (RD) framework [20], and taking advantage of a user navigation model to manage the client cache and to prefetch data [21].

When it comes to prefetching strategies for remote browsing of JPEG 2000 images, the work by Descampe et al. [19] provides a comprehensive view of the state of the art along with their own proposed solution. In their work, the authors propose and evaluate several solutions to anticipate future WOIs in order to achieve better responsiveness. However, all the solutions only take into consideration user exploration of single large-scale images, discarding the dynamic nature of navigation along an image sequence.

For the case studied in this chapter, the resolution of the images is not as high as that used by Descampe et al. as it is more important to improve the responsiveness to user movements along an image sequence as opposed to browsing just one single image. Moreover, the approaches proposed in [19] require special scheduling of JPEG 2000 packets, which is rather difficult to implement given the existing standard.

To achieve an acceptable level of responsiveness and avoid disturbing quality gaps while navigating through an image sequence, a special prefetching procedure is required. In the following sections, we present an efficient prefetching strategy for remote browsing of JPEG 2000 image sequences that offers good performance and smooth transitions, as well as easy implementation.

6.3 Context description

Client/server JPIP communication is based on the exchange of requests and responses. Within each request, clients specify, among other parameters, the remote file to explore and the WOI to be shown. Files may contain a sequence of N different images (in the case of JHelioviewer, they are related to a specific time range), so requests must also include the desired range of images $[a, b]$, with $0 \leq a \leq b \leq N-1$. Without any additional user interaction, the same WOI is retrieved from all the images within the time range.

There are two kinds of possible JPIP requests: stateless and session-oriented ones. Stateless requests are independent of each other, and no state is recorded during the exchange of messages between clients and servers. In the case of the session-oriented requests, all the requests related to the same remote image file are associated with the same session. This allows the server to remember which parts of the image have been sent to clients, thus avoiding redundant transmissions, e.g. for overlapping WOIs. Most of the remote browsing applications use this type of session-oriented communication.

By default, JPIP servers assume that clients always retain all the data received within a session. If a client needs to remove some of the retrieved data, for example to free up memory, the server should be notified. This study does not deal with client resource restrictions (e.g. memory), and we therefore assume that there are none.

Session-oriented communications allow clients to control the data flow. In the request of a certain WOI, a client can specify the maximum length L desired for the server response, then retrieve the data of the WOI in increments of L by simply repeating the same request several times. In the case of image sequences, the response data should be uniformly distributed by the server over the requested sequence.

For each request, the value L may be adapted depending on specific requirements. The most common scheme balances the usage of the available bandwidth and the response time to WOI changes by the user. Notice that larger values of L lengthen the elapsed time between the reception of data of two consecutive WOIs, especially at low bandwidths. This generates long response times for the user interaction. On the contrary, a faster response time is achieved when smaller values of L are used, even though additional overhead is generated due to the increase in the number of transmitted HTTP headers.

Some client applications, like kdu_show [7], adapt the L value according to the relation between the RTT (Round-Trip Time[1]) and the time T taken to extract the message

[1]The round-trip time is the elapsed time between the instant when the client generates the request and

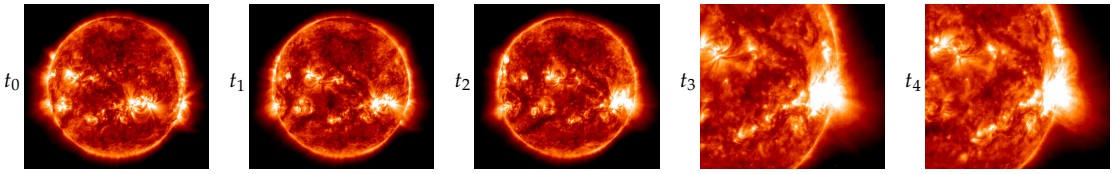

Figure 6.1: An example of different points in time of a remote browsing session. In the beginning, the user specifies a data source and a time range, and the first image is presented at time t_0. At time t_1 the image taken from the Sun has changed, due mainly to its rotation. At time t_2 the user specifies a WOI. t_3 and t_4 are points in time during the remainder of remote visualization session, where only the selected WOI has been transmitted and displayed.

data from the communication link, for each server response. Then, L is modified with the aim of equaling RTT$/T$ to a certain target value (L is decreased if the ratio is higher and increased if it is lower). This is done just after retrieving a server response and before performing the next request. In the approach proposed in this chapter, this method is implemented at the client-side.

This work focuses on JPIP applications, such as JHelioviewer, designed to explore remote image sequences. Figure 6.1 shows an example of five consecutive times during a remote browsing session using JHelioviewer. Once the user has selected a time range, the server builds a virtual JPEG 2000 file which only contains links to those images whose time stamp belongs to that time range. The client starts a JPIP session for that file and requests the first image, displayed at time t_0. The user can then watch the sequence of images belonging to the time range one by one. In this example, at time t_2 the user zooms in on a certain region, thus changing the current WOI. From this new WOI, the user continues moving forward through the sequence (times t_3 and t_4).

This type of interactive browsing requires a new communication scheme capable of offering smooth transitions while maintaining good responsiveness. It is also designed to be implemented over session-oriented JPIP communications. An added value is that this scheme is easy to implement on the client-side by simply combining the parameters L and $[a, b]$ of every request, and does not require any server or protocol modifications.

6.4 The proposed prefetching strategy

The method discussed in this section assumes that the images have been encoded using a suitable collection of encoding parameters. These parameters should allow spatial and quality scalability, and the definition of WOIs without any transcoding on the server side. It is also assumed that, for every WOI request, the JPIP server delivers the associated data minimizing the distortion of the displayed imagery.

the instant when the first bit of the reply arrives to the client.

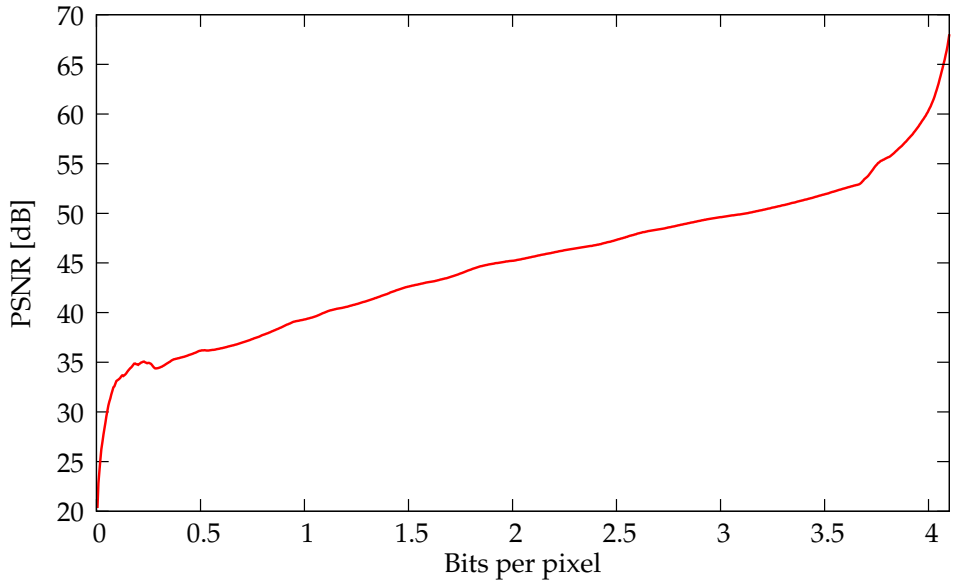

Figure 6.2: Rate-distortion curve for the SDO/AIA image *aia_test.lev1.304A_2010-06-12T09_29_26.13Z.image_lev1*. The reversible path of JPEG 2000 has been used.

As described in the previous section, a typical JHelioviewer user will spend some time displaying a concrete WOI (that could be the entire image) of a given single image and some time reproducing the entire image sequence. Frequently, they have to repeat these steps several times for the same image sequence (see Figure 6.1), pausing the movie mode at any image.

In the simplest transmission strategy (without prefetching), the data requested by the client belongs exclusively to the WOI of the currently displayed image. Therefore, the quality of a given WOI of a sequence will be proportional to the amount of time that WOI has been displayed and the available band-width of the transmission link during this time.

A rate-distortion curve of a typical image of the Sun (see Figure 6.2) indicates that, for a constant transmission rate, the visual quality of the images increases much faster in the beginning of the transmission. Therefore, in order to maximize the quality of the entire image sequence, at any time of the visualization, a given part of the bit budget should be dedicated to the currently displayed WOI (w) and the rest of the budget should be used for prefetching the same w of the subset of the remaining images of the time range.

To explain how our prefetching procedure was designed, some definitions are required. Let $E_w(b)$ be the distortion between w and $w^{(b)}$, where $w^{(b)}$ is the reconstructed

WOI after receiving b bits of w, calculated by means of the MSE (Mean Square Error)

$$E_w(b) = \text{MSE}(w, w^{(b)}) \ . \tag{6.1}$$

From our experience in this field we have observed that $E_w(b)$ exhibits an approximately exponential behavior (in this case with a negative exponent).

Thus, when b bits have been received for a given WOI w, the value of $E_w(b)$ can be used to decide for the next request which percentage of L is devoted to improving the current WOI (incrementing its quality i_c) and which percentage of L is used for prefetching other images from the time range.

A central issue related to the calculation of $E_w(b)$ is its dependence on the current content of w, i.e.; image data that are only known at the end of the transmission. However, taking into account the exponential trend of $E_w(b)$, this behavior is fairly similar to the behavior of a function that only considers the differential quality increments of $w^{(b)}$, i.e.;

$$dE_w^K(b) = \text{MSE}(w^{(b)}, w^{(b-K)}) \ ,$$

for a certain constant increment of received bits, K.

Note that the values of $dE_w^K(b)$ will decrease along the transmission of an image, but faster at the beginning of the transmission than at the end. Therefore, the value of $dE_w^K(b)$ can be useful to decide whether the data requested next should be part of the currently displayed image or should be dedicated to prefetching data from the other images within the time range. The idea is that, if the $dE_w^K(b)$ value is small enough, then most of the requested data should be dedicated to the time range prefetching.

With this idea in mind, the normalized percentage of each request to be dedicated to the prefetching can be modeled as

$$P_w^K(b) = \sigma e^{-dE_w^K(b)} \ , \tag{6.2}$$

where the value $0 \leq \sigma \leq 1$ is used to limit this percentage. Figure 6.3 shows the effect of K on the calculation of this percentage along the transmission of an image (with $\sigma = 1$ for all the cases and also for the rest of the figures of this document).

It can be seen that the larger the value of K is, the smaller the value of $P_w^K(b)$ and, therefore, the less aggressive the prefetching will be.

It should also be noticed that excessively small values of K (less that 50 bytes) could lead to the current and the next WOIs being identical, which will erroneously increase the prefetching of data, and that excessively large values of K will produce a slow prefetching scheme that, if the available band-width is also small, could disable the

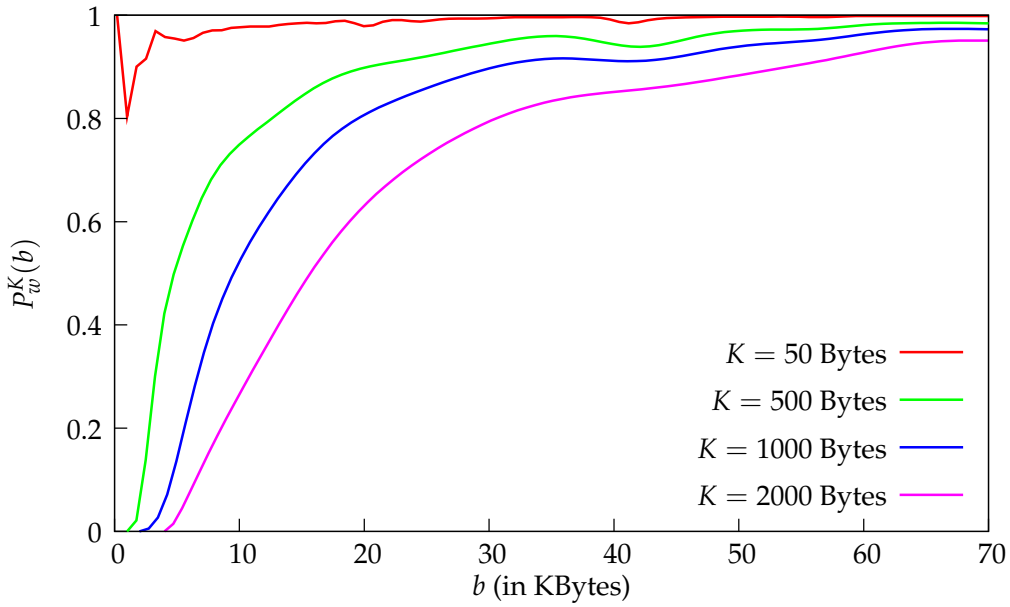

Figure 6.3: Impact of K on the prefetching model function $P_w^K(b)$.

prefetching of data altogether. Our experiments show that values close to 1000 bytes are suitable for a common remote browsing scenario.

According to $P_w^K(b)$, when the L value has been adjusted after receiving a server response, the next request to the server would be divided into

$$L_c = \big(1 - P_w^K(b)\big)L \tag{6.3}$$

bytes used for increasing the quality of the WOI within the current image (i_c), and

$$L_p = P_w^K(b)L \tag{6.4}$$

bytes used for prefetching the rest of the images of the time range. In our proposal, as the time range could potentially include too many images to be prefetched, the L_p budget is dedicated only to those images within the time range that are, at most, located λ images away from the current image i_c. Therefore, λ defines the size of a range of images $\{i_{c-\lambda}, \cdots, i_{c-1}, i_{c+1}, \cdots, i_{c+\lambda}\}$ that is centered on i_c but does not include it.

In order to assess the loss of quality of i_c due to the prefetching, we have depicted in Figure 6.4 a rate/distortion comparison, in terms of the Peak Signal-to-Noise Ratio (PSNR) in decibels, between a WOI of i_c retrieved without and with prefetching, for different values of K. This figure shows that, for all the tested values of K, the loss of quality produced by the time range prefetching is negligible at the beginning of the transmission, and visually insignificant at the end, which means that a user could hardly differentiate between both WOIs.

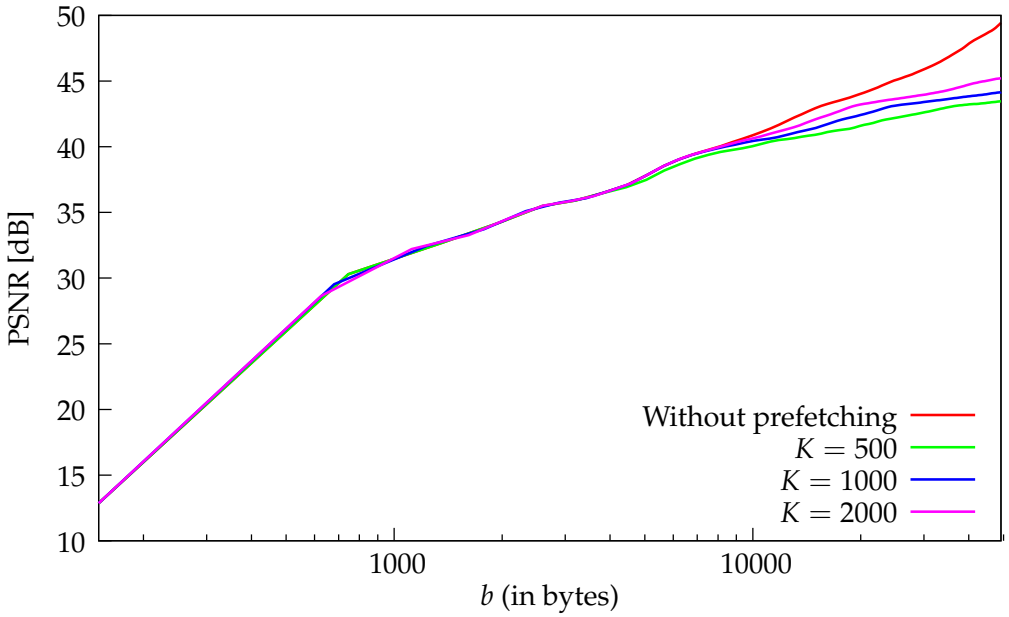

Figure 6.4: Quality of a window of interest of i_c (the currently displayed image) retrieved with and without prefetching.

Finally, from an implementation perspective, several points should be clarified. The JPIP does not allow the specification of different values of L within the same request. Due to this limitation, in order to carry out the prefetching of the time range, each request must be divided into two requests: one with L_c and another one for the prefetching with L_p. These requests should be sent continuously, profiting from the transmission pipelining, in order to avoid any communication delays.

Since an independent request must be used for i_c as well as for prefetching, it is necessary to control the overhead generated by the protocol. Experience leads to the assumption that, on average, JPIP servers include around $H = 300$ additional bytes within each response due to headers. Once the values L_c and L_p have been obtained, they must be modified depending on the ratio H/L_p. If this ratio is above a certain threshold, L_c is set to L, thus discarding the second prefetching request.

Experimental results show that a good value for this threshold is 0.5 or less. Changes in the value of H might also be taken into account during the communication for a better overall performance.

Once the current WOI has been completely received, the client should continue prefetching data using $L_p = L$. This makes it possible to also exploit the time spent by the user to analyze the content of the image.

6.5 Experimental evaluation

The approach proposed in this chapter has been implemented in the JHelioviewer client. A random user browsing session composed of 200 consecutive movements over a remote file that contains a sequence of eighty-eight 4096×4096 solar images has been generated. For each condition evaluated, the same session was simulated twice, with and without prefetching. The Kakadu JPIP server [7] has been used for these experiments. The client/server bandwidth has been fixed to 10^6 bits per second, with a RTT of 1 second.

A slightly modified version of the user model proposed by Descampe et al. has been used for generating the random user browsing session of the 200 movements. The possible user movements have been reduced to five: panning, zooming in, zooming out, moving forward and moving backward. The first movement consists of changing the position of the current WOI to a new random position within the same resolution level, with a distance of 128 pixels. After a panning movement, the WOI is fully overlapped by the precinct partition (128×128 for every resolution level) without gaps.

The zooming movements change the resolution level of the WOI one by one. The zooming-in movement is limited so that the minimum resolution level allowed is 512×512. The last two movement types allow moving through the sequence one image at a time. None of these movements modify the WOI dimension, which is always 512×512 pixels.

It is assumed that the user behavior is defined by a first order Markov process, so given a movement of one type, the probability of choosing the same movement as the next one is δ, while the probability of choosing a different one is $(1 - \delta)/4$. The experiments in this study have used a value of $\delta = 0.4$. From the point of view of these experiments, the value of δ is not critical for evaluating our proposal since the distribution of movements is homogeneous. In Descampe's work this value had to be evaluated because it was associated to the predictability of the user behavior, a factor that affected the prefetching scheduler, which is not the case for our solution. We have therefore chosen an intermediate value for δ, corresponding to a user behavior between almost deterministic ($\delta = 0.9$) and fully random ($\delta = 0.2$).

During the simulated browsing session, as soon as the quality of the current reconstructed WOI achieves a quality better than a certain PSNR threshold Θ, the next movement is triggered after a reaction delay B_τ, expressed in bytes. As in Descampe's work, we have evaluated the values 30, 35 and 40 for Θ. For all values of Θ we have assumed $B_\tau = 0$, with the exception of $\Theta = 40$, for which we have also used the values 10 and 20 kilobytes.

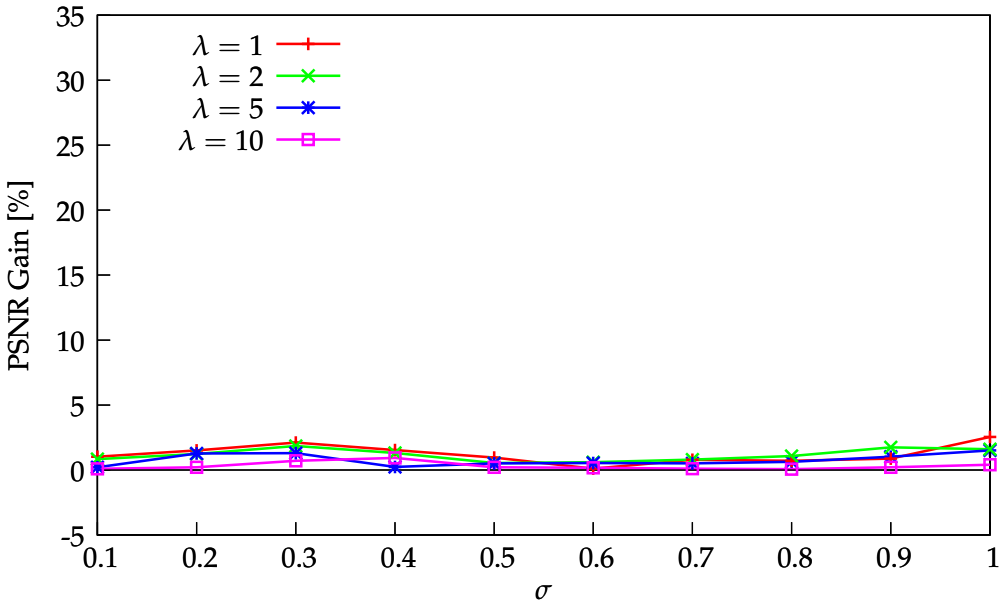

Figure 6.5: Experimental results for $\Theta = 30$ dB and $B_\tau = 0$ bytes.

A total of 40 different conditions have been evaluated, varying σ from 0.1 to 1, in steps of 0.1, with values of 1, 2, 5 and 10 for λ. The average difference of the PSNR obtained in the remote browsing session, with and without prefetching, has been calculated for each condition. This difference is expressed as a percentage relative to the first session. The value used for K was 1000 bytes.

Figure 6.5, 6.6 and 6.7 show that the best results are obtained for $\Theta = 40$. For $\Theta = 30$ the improvement is hardly noticeable because the WOI is moved before a significant stabilization of the differential quality has happened, and therefore prefetching cannot be applied before the next user movement.

Figure 6.7 shows that with our solution there is always an improvement in the average value of the PSNR, independently of the values used for σ and λ. Nevertheless, the maximum improvement is achieved around $\sigma = 0.9$.

In all cases, the higher the value of λ, the less improvement is achieved. Taking into account the chosen user model, with one-by-one movements through the image sequence, this result is to be expected. It is perceivable that with another user model, with different degrees of freedom of movement through the sequence (e.g.; allowing the user to skip images), the impact of λ might be different.

Taking into account that the best results have been obtained for $\Theta = 40$, we have also carried out more experiments for evaluating the effect of using this value of the threshold 8($\Theta = 40$) with two different values of the reaction delays, $B_\tau = 10$ and $B_\tau = 20$, in kilobytes, as shown in Figures 6.8 and 6.9, respectively. It is observed that the performance worsens with increasing value of B_τ, and it is even possible to obtain

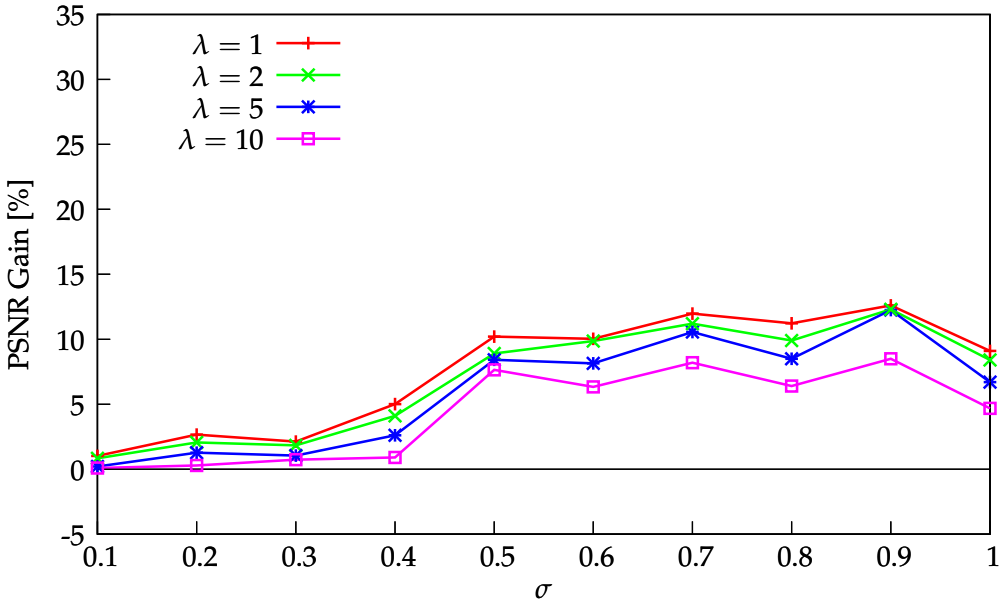

Figure 6.6: Experimental results for $\Theta = 35$ dB and $B_\tau = 0$ bytes.

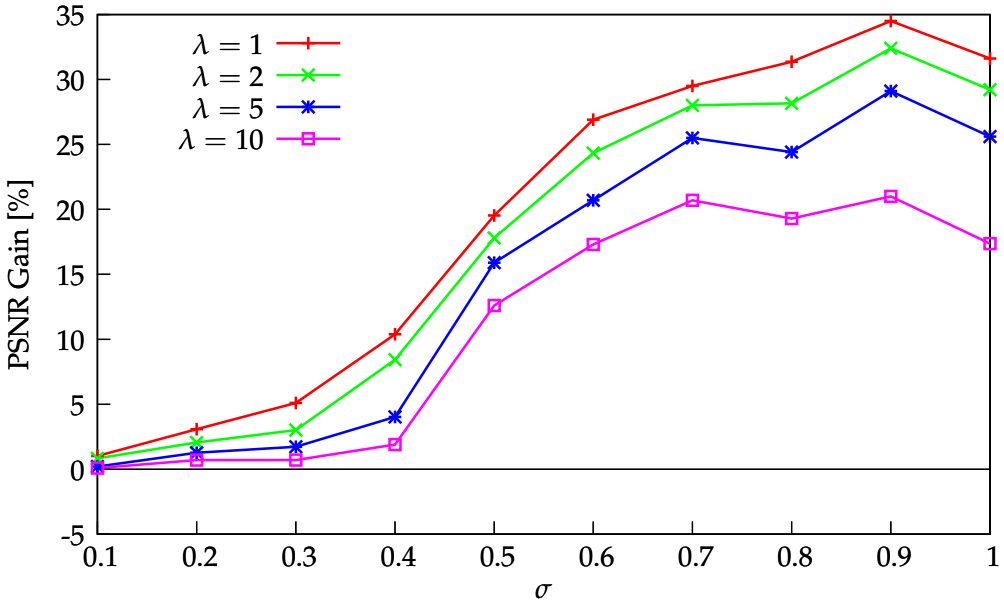

Figure 6.7: Experimental results for $\Theta = 40$ dB and $B_\tau = 0$ bytes.

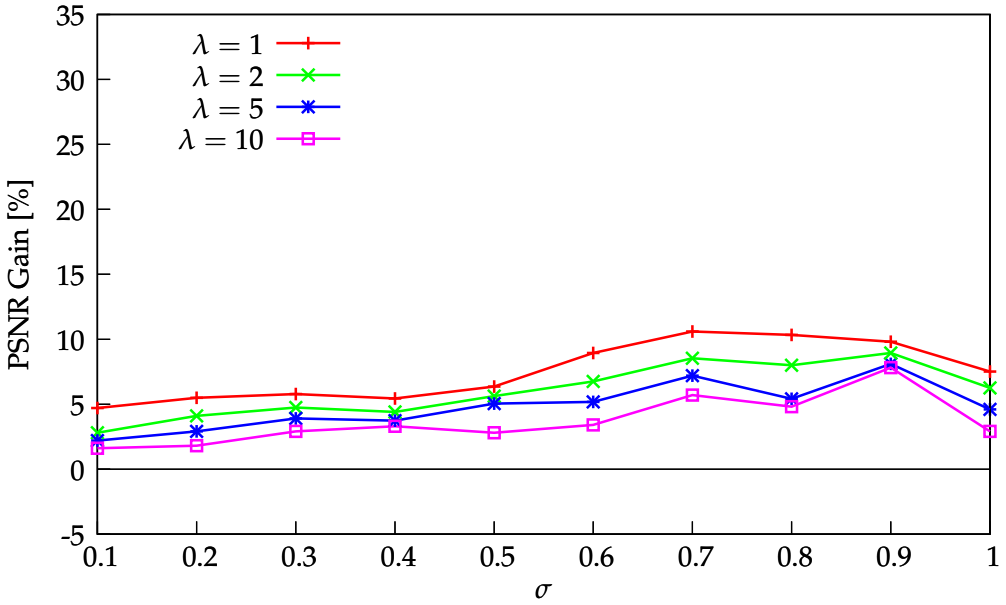

Figure 6.8: Experimental results for $\Theta = 40$ dB and $B_\tau = 10$ kilobytes.

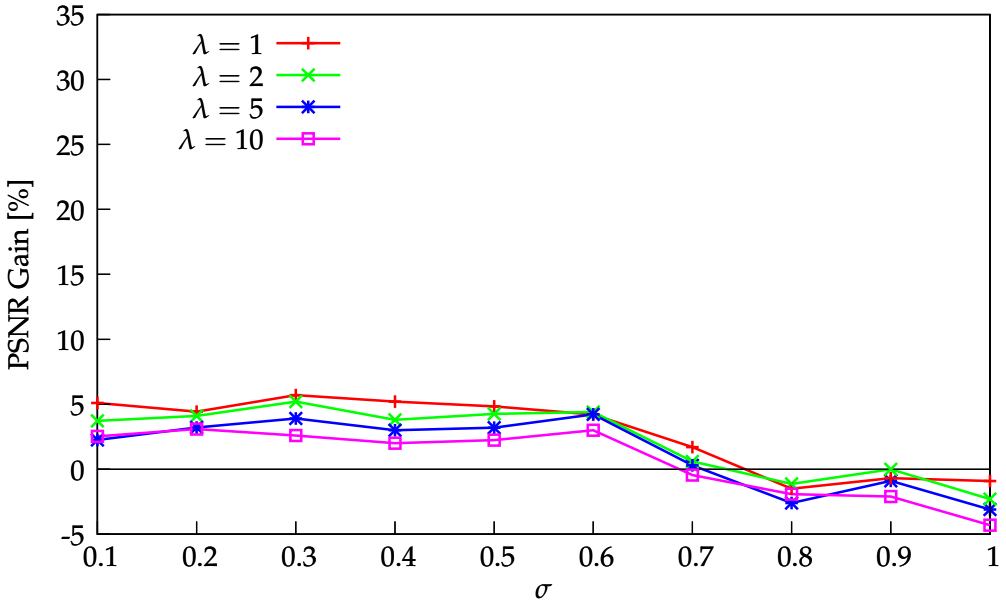

Figure 6.9: Experimental results for $\Theta = 40$ dB and $B_\tau = 20$ kilobytes.

worse PSNR values for large values of σ. These negative values are achieved due to the PSNR metric. Over 40 dB, the differences in the visual quality of the Sun images is hardly noticeable. At these high σ values, the client bandwidth is therefore almost completely dedicated to prefetching. The PSNR is thus incremented very slowly, while the non-prefetching solution continuously increases the PSNR value. Although this difference cannot be noticed by the user, the exact results are affected.

The results presented in this chapter show that the proposed solution improves the general user experience, defined in terms of PSNR, when browsing remote sequences of JPEG 2000 images. The best value for σ is associated to the speed of the user movements: for fast movements, the best results are obtained for large σ values; for slow movements, it is better to use small σ values that guarantee prefetching without significantly affecting the download of the current WOI.

The client application might even adjust the value of σ dynamically depending on the user behavior. When the user is moving through a sequence looking for a specific image, the movements are usually quite fast. However, once the user has located an interesting image, the movements become slow, with high reaction delays.

CHAPTER 7

The ESA JPIP server

7.1 Introduction

The client side of the JHelioviewer project (see Section 1.2) is being developed in Java, whose code is stored under a open-source license in Launchpad. It uses the Kakadu JPEG 2000 library, developed by D. Taubman [7]. This library is currently one of the best and most used JPEG 2000 implementations because it is highly optimized. Moreover, although it is a commercial software, its binaries can be redistributed without restrictions for open-source solutions.

The Kakadu package also contains some demo applications, including a completely functional JPIP server. Even though this server has been improved significantly throughout all the library versions, it still suffers from scalability and stability restrictions. In the case of the JHelioviewer project, where a substantial load of data transmission and client connections are expected, the solution provided by the Kakadu library does not offer enough performance for very large imagery systems, as is shown later in this chapter.

Although there are other freely available implementations of the JPIP, none of them is capable of complying with the necessary requirements imposed by the JHelioviewer project. One of them is the open-source OpenJPEG JPEG 2000 library which was developed under the 2KAN project [11]. In this library there is an implementation of a JPIP server, called OpenJPIP [9], but unfortunately, for the time being, it only supports tile-based streaming, which is only recommendable for certain specific applications, not for the case of the JHelioviewer project. Most importantly, OpenJPIP does not implement a full server architecture, like the Kakadu server. It is designed as a CGI (Common Gateway Interface) module for an existing Web server. This implies that the development does not tackle the server performance or scalability at all, depending these issues on the base system used. These restrictions led it to be discarded as well for the JHelioviewer project. Another interesting implementation is the CADI software [2] developed by the Group on Interactive Coding of Images (GICI) at the Universitat Au-

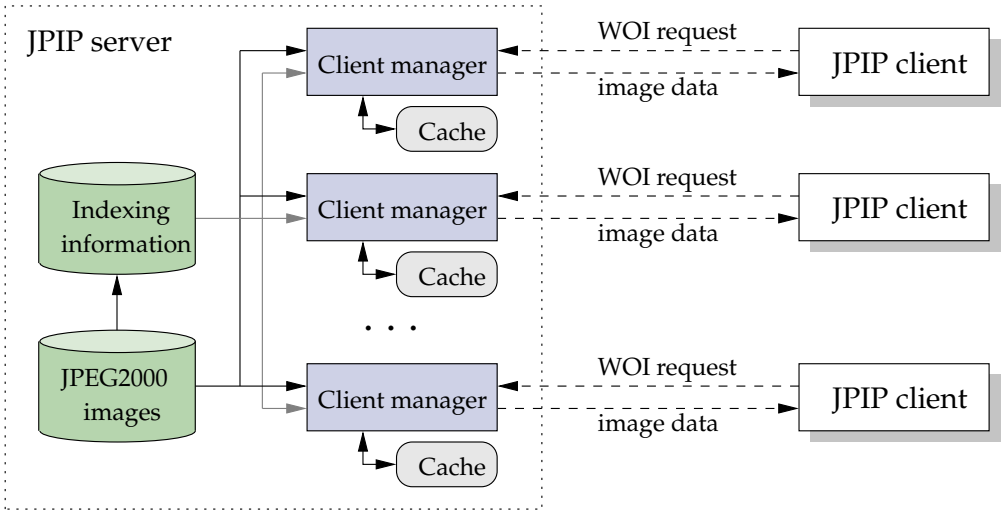

Figure 7.1: Common functional architecture of a JPIP communication system, focused on the server side.

tonoma de Barcelona. In this case, the approach was discarded because it is written in Java, a programming language that is not as efficient as C++ to control the CPU and the memory usages of the server host.

In this context, where JPIP server implementations capable of complying with the JHelioviewer project requirements were missing, ESA decided to finance the development of a new efficient and scalable open-source JPIP server. This chapter analyses and assesses the approach we have developed for this application, whose first version is currently available on Launchpad [3].

7.2 Server architectures

Figure 7.1 shows the common functional architecture of a JPIP communication system, focused on the server side. For each client connection, a new communication session is established and handled by a client manager. Although the JPIP protocol allows stateless connections, they are not commonly used.

Within the session a client can open different channels, one for each remote image to explore. Clients explore the desired images by means of WOI requests. A WOI is usually identified by a rectangular region and a zoom or resolution level. The client manager extracts from the associated JPEG 2000 image those parts related to the WOI and sends them to the client.

The client usually imposes a length limit for the server responses with the aim of controlling the communication flow. The complete response to a WOI is thus completed in several different message exchanges, repeating the same request several times. This is possible thanks to the cache model maintained by each client manager,

which records which image parts have already been sent.

For this data extraction the server needs an indexing information that is generated by parsing the image files. The way this information is generated has an enormous effect on the server performance. For example, for each WOI request, it is always feasible to parse the entire image file looking for the required parts. This does not consume memory, but it involves considerable processing and disk load. A completely different approach is to pre-build a complete index file and load it completely before attending a WOI request of an image. This reduces the processing and disk load as much as possible, but consumes too much memory.

A hybrid approach achieves a good relation/compromise between the memory consumption and the CPU/disk usage. The process consists of pre-building some small indexing files, which contain the references of the main parts of the image, and then parsing the images on demand, depending on the client's requests.

The implementation of a JPIP server must consider that the indexing information is shared by all the client managers. Depending on how it is generated, the sharing mechanism may or may not become very complex or not. In the case of complete pre-build index, the access of the client managers is only for reading. However, for the hybrid approach, the client managers access to the index information is for reading as well as for writing.

At the moment of writing this thesis, there is no published work related to either JPIP server implementations or architectures. This is why it has not been possible to include any reference nor comparison to previous related works. Nevertheless, the architecture of a JPIP server is quite similar to that of a common Web server. For example, as was mentioned in the introduction, the OpenJPIP server has been implemented as a layer for a Web server. Next some existing works related to Web server architectures are analyzed.

There are multiple possible approaches for implementing a Web server, the multi-processes (MP) and multi-threads (MT) are the most common ones [14]. With MP, each client is handled by a different process. The main advantage of this approach is the stability: if one process crashes, the other ones are not affected at all. On the contrary, this solution achieves a lower performance than MT in terms of memory consumption, operating system load (creating and killing processes) and inter-processes communication. The mechanisms for sharing information between processes in the existing platforms are usually less efficient than in the case of threads. The most commonly- used Web server nowadays, Apache [1], adopts the MP approach in the Unix version 1.3 and in the version 2.0's multiprocessing (MPM) pre-fork module.

The MT approach provides an easy and natural method of programming a server;

the sharing and communication between threads becomes simple and efficient. However, it lacks in stability because if a thread crashes, the entire server process breaks down, stopping all the other threads as well. The Kakadu server [7], for example, adopts this approach.

With the aim of achieving a balance between the two previous solutions, some implementations use an hybrid approach (MP+MT), dividing the server in several processes, and each process in several threads. This increases the stability and obtains a performance near the MT solution. The Apache 2.0 Worker MPM implements it. The main drawback of this approach is inherited from the MP, that is, the sharing and communication between the processes, and hence between the threads of different processes.

Apart from the MP, MT or MP+MT, there are many other different proposals studied by the research community, some of them have been compared in terms of performance in the work of D. Pariag et al. [63]. A particular and frequently-referenced proposal is the Flash server of V.S. Pai et al. [62]. It implements an AMPED (Asynchronous Multi-Process Event Driven) architecture that avoids the use of blocking I/O operations, and hence reducing the associated idle times. Although it showed promising results, the work of Gyu Sang Choi et al. [18] demonstrated that compared to an MT approach it suffers from scalable performance on multi-processor machines.

7.3 Proposal description

The open source project described here was carried out with the aim of implementing, using the C++ programming language, an efficient and highly stable/scalable approach for a JPIP server. It has been specifically designed for Unix systems in order to fully profit from its characteristics, discarding a portable design which might compromise the efficiency.

The ESA JPIP server is capable of handling the following JPEG 2000 image files: raw J2C, JP2 and JPX with or without hyperlinks. The main requirement for the image files is that they must contain PLT markers, defined in the standard, with the information about the length of all the packets. These markers allow the server to build the indexing information of the different parts of the image without decoding. It simplifies the code and avoids using any JPEG 2000 engine.

Almost any type of packet progression is allowed for compressing the images, but the RPCL progression is strongly recommended for achieving an efficient performance because of the organization of the packets in the file. Other implementations, like Kakadu, recommend this progression as well.

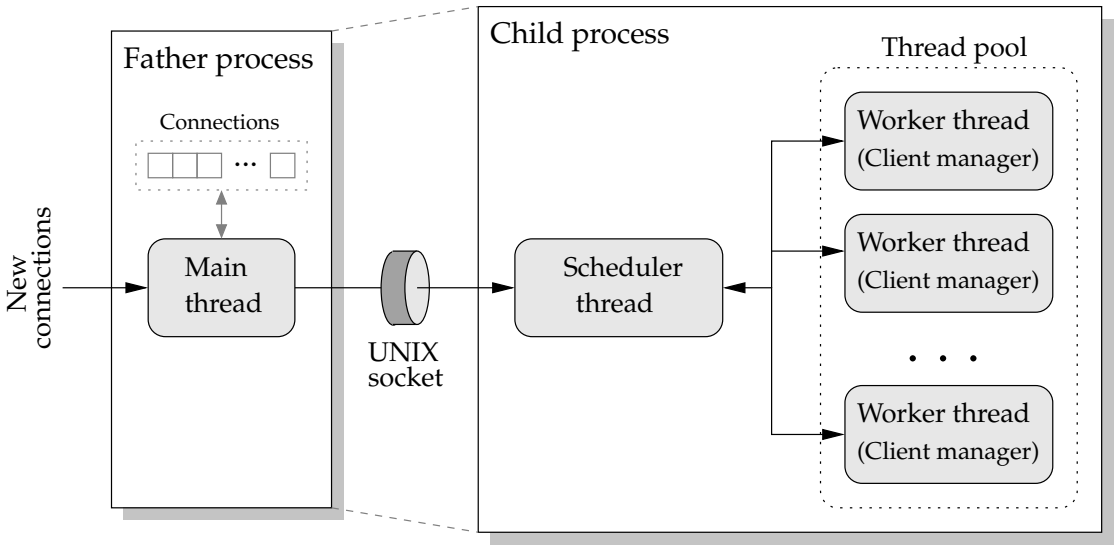

Figure 7.2: Schematic representation of the ESA JPIP server architecture.

Figure 7.2 shows a schematic representation of the server architecture. It consists of a hybrid model combining both approaches: process and thread. This architecture is not a classic hybrid MP+MT model though, as it is implemented in Apache, for example. Nevertheless, it is more of a pure MT approach with minimum MP support for achieving good robustness. There are only two processes, hereinafter called father and child. The child process maintains all the working threads. The father process creates and watches the child process by forking. It also possesses the listening socket of the server to accept new incoming connections. When a new client connection is established by the father process, it sends this connection to the child process through a UNIX-domain socket and records it in a vector where all the opened connections are recorded as well. If the father detects that the child has finished (e.g. due to a crash) it creates a new child process by forking itself. Taking into account that it inherits the vector of the current opened connections, it can continue handling them without interruptions for the clients.

The child process provides all the functionalities to handle the client connections. It contains a scheduler thread for reading the new connections sent by the father through the UNIX-domain socket. The scheduler thread assigns each connection to a working thread available in the maintained thread pool.

Each working thread implements the necessary functions, explained in Chapter 3, associated to the client manager module shown in Figure 7.1. The indexing information can easily be shared in the memory by all the threads without the efficiency restrictions when dealing with processes.

In order to generate the indexing information of the images, a hybrid approach has

been adopted. The first time an image is requested from the server, a small associated cache file is created containing the index of the main parts of the image, mainly the position of the header and the PLT markers. This cache file is loaded by the server whenever the same image is requested again.

With the help of this cache file, the indexing information is generated by the server on demand depending on which regions are explored by the clients. Actually, the more resolution levels of an image the user explores, the larger the related index data becomes. The space required for this data has been reduced considerably, requiring the minimum possible number of bytes for each index item.

The index of each opened image to be served is stored as a node in a double-linked list shared by all the threads. Each node may also have references to other nodes of the list. For instance, in the case of a JPX file with hyperlinks, it is represented by a node which points to a set of other nodes, each one associated to each hyperlink of the file.

The access control of the threads to this shared information has been implemented using two different mechanisms. A general mutex locking mechanism has been adopted for reading/modifying the list. These operations are fast and only performed when opening/closing images. For each list-node, a reader/writer locking mechanism has been used in order to control the access to the indexing information of each image. This mechanism gives a higher priority to the readers than to the writers. The server behavior profits from this mechanism because the read operation is the most common, while the write operation is performed only when incrementing the indexing information for a new resolution level. All of these locking mechanisms are available by means of the POSIX thread library.

This architecture provides a fault-tolerant and robust approach for the server, as well as offering good performance. The multi-threading solution implemented in the child process is efficient in terms of memory consumption and fast sharing/locking mechanisms. Having separated the client handling code from the father process provides robustness and security. If the child process crashes, the father process will be able to launch a new child process, maintaining all the opened client connections operational (clients do not perceive any problem).

The image data is transmitted efficiently. The precincts that are geometrically overlapped by the requested WOI are always sent first, following the LRCP progression. This is the optimal alternative in terms of rate/distortion, as was analyzed in Section 2.5. As will be shown in the following section, this achieves a significant gain in terms of rate/distortion.

	Memory (MB)		CPU (%)	
	Average	Deviation	Average	Deviation
ESA server	30.17	1.77	213.25	76.05
Kakadu server	1871.46	345.56	176.54	128.04

Table 7.1: Results of the benchmarking.

7.4 Evaluation

The approach proposed in this chapter has been experimentally tested. Performance results have been compared to the JPIP server provided by the commercial Kakadu package [7]. Currently, this package is the most referenced JPEG 2000 implementation, due to its good performance.

The aim of the first test was to compare both approaches in terms of memory consumption and CPU usage. 20 linked JPX files were created with 1000 different frames, each one corresponding to 4096×4096 SDO Sun images. Every 5 seconds a flash crowd of 100 connections was established. Each connection is related to a JPX image from the available set (20) and simulates a client that plays the video for 30 seconds, requesting 15 sequential images in each query and exploring all 1000 frames. After 30 seconds, all connections are closed, releasing the related channels as well. The WOI used was $(0, 0, 1024, 1024, 2)$. This scenario was running for one a week.

Although there are not many simultaneous clients in this experimental scenario, the server load is quite high, due to the large amount of image data that needs to be handled and distributed among many different files. Moreover, this scenario is a common situation within the context of the JHelioviewer application.

When generating the JPEG 2000 images used in this experiment, the PLT markers have been included and the following compression parameters have been used: 8 quality layers, 8 resolution levels and RPCL as progression order. Precincts have been used with a size of 128×128.

Table 7.1 shows the average and standard deviation of the experimental results obtained from this test. As can be seen, the Kakadu server needs around 2 gigabytes of RAM, while the ESA server only needs 30 megabytes. Moreover, the standard deviation says that the memory consumption of the Kakadu server is much more variable than the ESA server, which, on the contrary, maintains the same memory consumption almost all the time.

The CPU usage is similar in the two solutions, although the Kakadu server seems to need less. Notwithstanding, the logs have shown many records with 0 usage. Con-

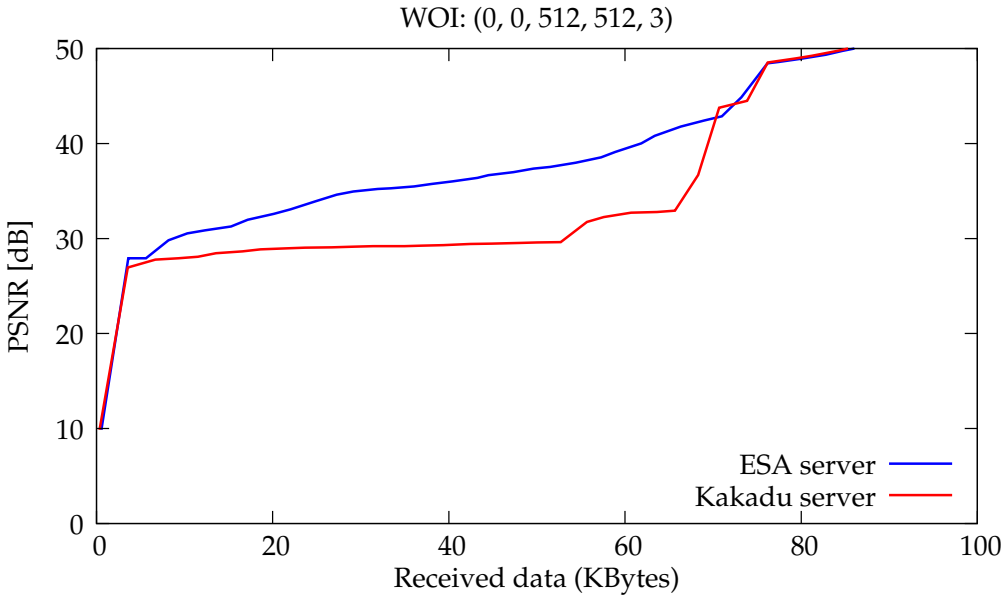

Figure 7.3: Comparisons, in terms of PSNR vs. data received, between the Kakadu server and the ESA server for the first WOI movement.

sidering that the logs have been recorded every 5 seconds, this only means that the Kakadu server generates delays in function of the connections. This would be coherent with the differences in the standard deviation results.

The average throughput, in terms of responses per second, has also been recorded in this scenario, when all the 100 clients are communicating. The Kakadu server achieves a value of around 232, while the ESA server raises up to 1068.

In the context of the remote browsing systems, where a JPIP server is located, it is also very interesting to evaluate the quality of the reconstruction of the WOI provided, measured in terms of the PSNR[dB] versus the amount of data received at the client side. This measurement is related to the user experience because the user always wants to see the best quality as soon as possible.

The sequence of WOIs $(x, y, width, height, res.level)$ that has been used in this experiment is: $(0, 0, 512, 512, 3)$, $(512, 512, 512, 512, 2)$, $(1024, 1024, 512, 512, 1)$, $(2048, 2048, 512, 512, 0)$. It corresponds to a common user sequence using the JHelioviewer application and zooming in on a corner. Figure 7.3, 7.4, 7.5 and 7.6 show the rate-distortion curve generated by the ESA and Kakadu servers.

As can be observed, the ESA server provided better results. This is a direct consequence of the way it transmits the image data as explained in the previous section. The Kakadu server seems to use a different packet progression.

In the most rigorous scenario where the ESA server was tested, it was able to handle 1500 simultaneous connections, serving a total of 1500000 image files in parallel. The Kakadu server is not able to support this load.

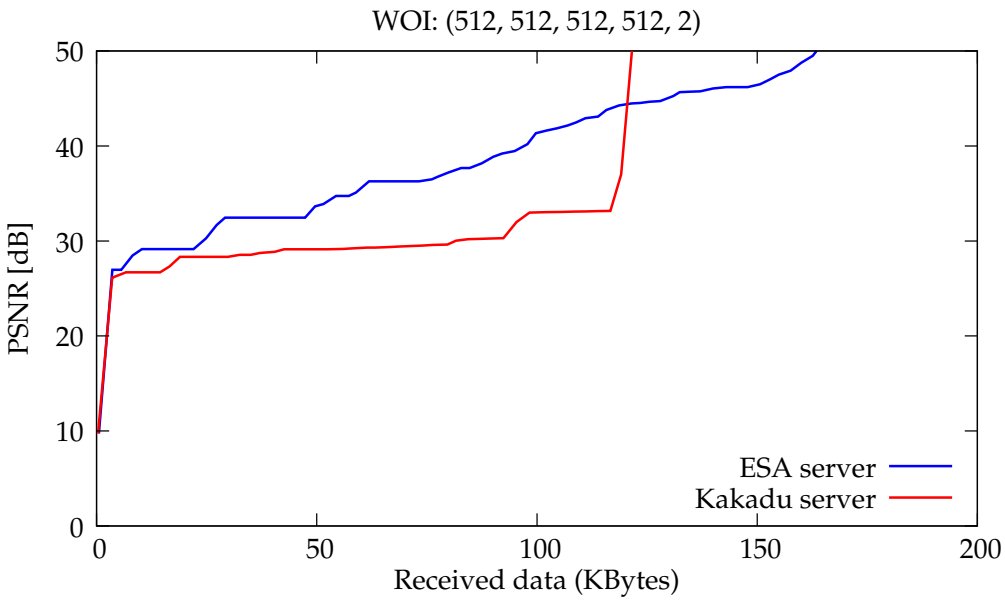

Figure 7.4: Comparison, in terms of PSNR vs. data received, between the Kakadu server and the ESA server for the second WOI movement.

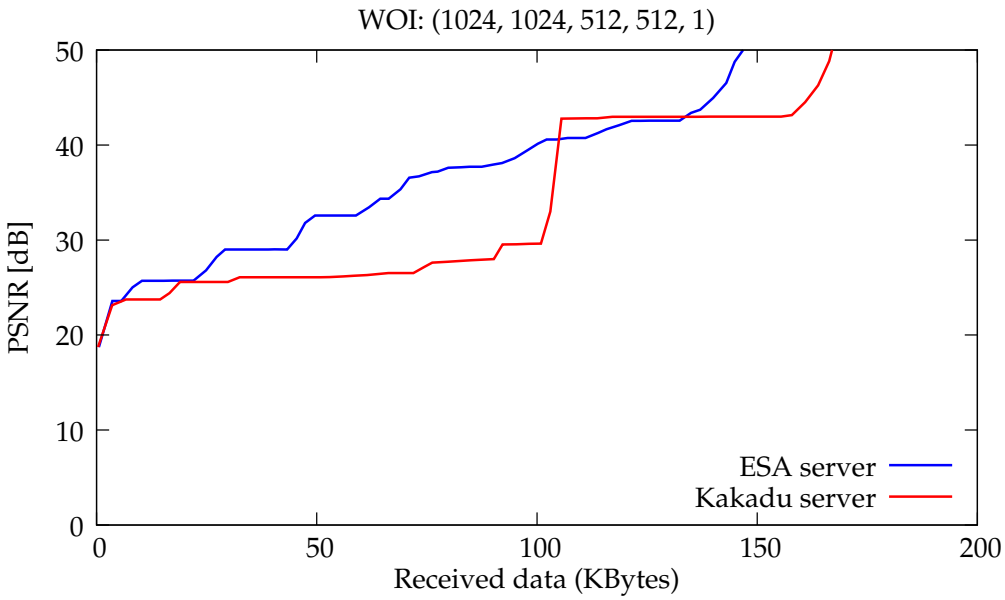

Figure 7.5: Comparison, in terms of PSNR vs. data received, between the Kakadu server and the ESA server for the third WOI movement.

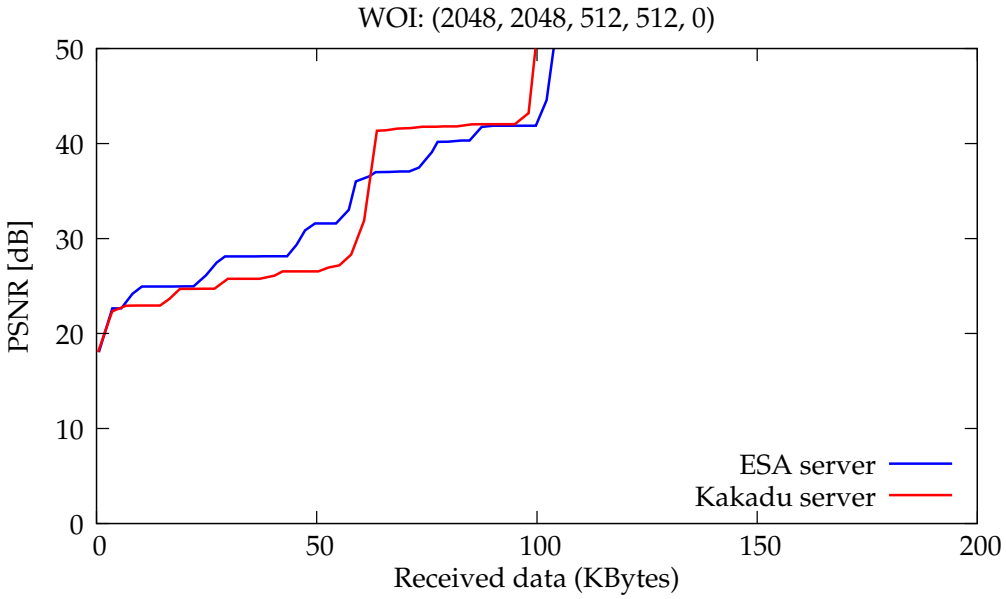

Figure 7.6: Comparison, in terms of PSNR vs. data received, between the Kakadu server and the ESA server for the forth WOI movement.

As it can be observed in the evaluation, the ESA JPIP server offers better results, in terms of memory consumption and CPU usage, than the Kakadu server. Moreover, at least in the carried out experiments, it seems to offer also better values in terms of rate-distortion. This comparison, and with the fact of being an open-source implementation, makes the ESA JPIP server to be currently one best solutions for remote browsing applications using JPIP.

CHAPTER 8

Conclusions and future work

This thesis has been focused on the study and analysis of the JPEG 2000 standard in combination with the JPIP protocol in the context of the remote browsing applications. Scientific and technological results of this work consist of four main contributions for improving different aspects of this technology and fixing also some of its lacks.

The first contribution has been the proposal of the JPL format [36, 32] and the required architecture to exploit it, so that it is possible to implement an efficient remote browsing application for JPEG 2000 images without using the JPIP protocol, just by mean of a common HTTP/1.1 server. A full implementation was carried out, which is on-line available licensed as open source [6].

JPIP-W, the second contribution [34, 33, 35, 40], is an extension that improves the JPIP protocol for interactive Internet JPEG 2000 image browsing. JPIP-W takes advantage of the Web proxy infrastructure to reduce image retrieval transmission time, user latency and network traffic. JPIP-W uses a server-client model similar to JPIP, but when the JPIP-W server receives a WOI request, it sends a message that can be cached by Web proxies. Therefore, a request for an overlapped WOI is partly served by proxies. JPIP-W is not an alternative to JPIP; instead, it is a totally compatible add-on. The JPIP-W server can also work as a JPIP server if required. Experimental results show that JPIP-W outperforms JPIP when there is cached data. Although the initial JPIP-W delay is a little bit longer than that of JPIP, the benefits of being able to use Web proxies are clear throughout the transmission. Furthermore, when there is no cached data, JPIP-W efficiency is virtually as good as that of JPIP.

In this thesis, a new efficient prefetching technique for interactive remote browsing of JPEG 2000 image sequences has been also proposed. Its most relevant characteristics are: i) it offers an easy implementation that can be added to any existing JPIP client/server architecture; ii) from the client/server bandwidth available, a certain fraction is allocated to prefetching, which is estimated using a differential quality model function; and iii) an average improvement of the reconstructed WOI is always achieved, independently of how much fine-tuning is carried out.

Finally, the experience obtained during the research work led me to carry out a new implementation of a JPIP server for the JHelioviewer project [3]. The evaluation results show that the ESA JPIP server is better than the server provided by the Kakadu package, in terms of scalability (memory consumption and CPU usage) as well as in terms of rate-distortion. Its open-source license allows it to be used, maintained and improved freely by the Internet community. Therefore, this development is currently one of the best options for all systems in which a JPIP server is required [31].

There are many different possibilities for future works as a continuation of this work:

1. The JPIP-W protocol might be implemented in the ESA JPIP server, offering thus an alternative for reducing the server load profiting from the Web proxies.

2. In the case of the prefetching technique, an extension of it which utilizes sequences of images for video streaming would be very interesting to analyze. The ESA JPIP server offers a good point for researching and experimenting. It might be extended with different improvements related to the video browsing, or even with the peer-to-peer communication.

3. In the context of remote image/video retrieval, there is also room for working in a quite promising technique called *Conditional Replenishment*. Using this, a client could improve the quality of the reconstruction of a sequence of images that are temporally correlated (such as in a typical video), requesting to the server only those regions of the video sequence that are most relevant or that are more difficult to predict. Furthermore, this technique can be carried out only by the client-side of the network architecture, which means that it should be very scalable and that all the proposed solutions for the servers should be compatible with it.

Bibliography

[1] Apache HTTP server project. http://httpd.apache.org.

[2] CADI software. http://gici.uab.es/CADI.

[3] ESA JPIP Server. https://launchpad.net/esajpip.

[4] Google Earth. http://earth.google.com.

[5] JHelioviewer project. https://launchpad.net/jhelioviewer.

[6] JP2Viewer. https://launchpad.net/jp2viewer.

[7] Kakadu JPEG 2000 SDK. http://www.kakadusoftware.com.

[8] Listado de tipos MIME registrados por la IANA (Internet Assigned Numbers Authority). http://www.iana.org/assignments/media-types/index.html.

[9] OpenJPIP. http://code.google.com/p/openjpeg/wiki/JPIP.

[10] Squid Web Proxy Cache. http://www.squid-cache.org.

[11] The 2KAN Project. http://www.2kan.org.

[12] Joint Photographic Experts Group (JPEG). http://www.jpeg.org, October 2006.

[13] M.D. Adams and F. Kossentini. JasPer: A software-based JPEG-2000 codec implementation. In *IEEE Int. Conf. Image Processing*, volume II, pages 53–56, Vancouver, Canada, September 2000.

[14] D. Carrera, V. Beltran, J. Torres, and E. Ayguade. A hybrid Web server architecture for e-commerce applications. In *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on*, volume 1, pages 182–188, 2005.

[15] U. Catalyurek, M.D. Beynon, C. Chang, T. Kurc, A. Sussman, and Joel Saltz. The virtual microscope. *IEEE Transactions on Information Technology in Biomedicine*, 7(4):230–248, 2003.

[16] Y. Chae, K. Guo, M.M. Buddhikot, S. Suri, and E.W. Zegura. Silo, rainbow, and caching token: schemes for scalable, fault tolerant stream caching. *IEEE Journal on Selected Areas in Communications*, 20(7):1328–1344, 2002.

[17] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A Hierarchical Internet Object Cache. In *Proceedings of the 1996 USENIX technical conference*, pages 153–163, 1995.

[18] G.S. Choi, J.H. Kim, D. Ersoz, and C.R. Das. A multi-threaded PIPELINED Web server architecture for SMP/SoC machines. In *Proceedings of the 14th international conference on World Wide Web*, pages 730–739, New York, NY, USA, 2005. ACM.

[19] A. Descampe, C. De Vleeschouwer, M. Iregui, B. Macq, and F. Marques. Prefetching and caching strategies for remote and interactive browsing of JPEG 2000 Images. *IEEE Transactions on Image Processing*, 16(5):1339–1354, May 2007.

[20] A. Descampe, C. De Vleeschouwer, M. Iregui, B. Macq, F. Marqués, Bâtiment Stévin, and Place Du Levant. Pre-fetching and caching strategies for remote interactive browsing of JPEG 2000 images. In *Tech. Rep., TELE - UCL*, 2005.

[21] A. Descampe, J. Ou, P. Chevalier, and B. Macq. Data prefetching for smooth navigation of large scale JPEG 2000 images. *IEEE International Conference on Multimedia and Expo*, pages 908–911, 2005.

[22] S. Deshpande and W. Zeng. HTTP Streaming of JPEG 2000 Images. *International Conference on Information Technology: Coding and Computing*, pages 15–19, 2001.

[23] S. Deshpande and W. Zeng. Scalable streaming of JPEG2000 images using hypertext transfer protocol. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, pages 372–381, New York, NY, USA, 2001. ACM.

[24] R. Dosselmann and X.D. Yang. Existing and emerging image quality metrics. In *IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1906–1913, May 2005.

[25] EPFL, C. R. France, and Ericsson. Java implementation of JPEG2000, 2006.

[26] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *IEEE/ACM Transactions on Networking*, pages 254–265, 1998.

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC2068, January 1997.

[28] J.P. García-Ortiz. Google Summer of Code - JPEG2000 JPIP Server and Viewer Applet. http://dltj.org/article/gsoc-jpip/, August 2006.

[29] J.P. García-Ortiz, J.M. Dana, V. González Ruiz, and I. García. Broadcasting of H.264/SVC video over BitTorrent-like networks. In A. Dapena and J. Oliver, editors, *Actas del Workshop on Multimedia Data Coding and Transmission (WMDCT)*, pages 41–46. Garceta Grupo Editorial, September 2010.

[30] J.P. García-Ortiz, V. González-Ruiz, I. García, D. Müller, and G. Dimitoglou. Interactive Browsing of Remote JPEG 2000 Image Sequences. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3179–3182, 2010.

[31] J.P. García-Ortiz, C. Martin, V. González-Ruiz, J.J. Sánchez-Hernández, I. García, and D Müeller. Efficient and scalable open-source JPIP server for streaming of large volumes of image data. In *IEEE International Conference on Consumer Electronics*, pages 380–384, Berlin, 2011.

[32] J.P. García-Ortiz, V.G. Ruiz, and I. García. An efficient technique for remote browsing of JPEG2000 images on the Web. In *Actas de las XV Jornadas de Paralelismo. Computación de Altas Prestaciones*, pages 322–327, September 2004.

[33] J.P. García-Ortiz, V.G. Ruiz, and I. García. Improved JPIP protocol with proxy caching. In *Actas de las XV Jornadas de Paralelismo. Computación de Altas Prestaciones*, pages 328–333, Almería, España, September 2004.

[34] J.P. García-Ortiz, V.G. Ruiz, and I. García. Improving the remote browsing of JPEG2000 images on the Web. In *IASTED International Conference on Visualization, Imaging, and Image Processing*, pages 890–895, September 2004.

[35] J.P. García-Ortiz, V.G. Ruiz, and I. García. Improving Web proxy caching on browsing JPEG2000 remote images with JPIP. *WSEAS Transactions on Information Science and Application*, 1:167–172, July 2004.

[36] J.P. García-Ortiz, V.G. Ruiz, and I. García. Remote browsing of JPEG2000 images on the Web: evaluation of existing techniques and a new proposal. In *IASTED International Conference on Visualization, Imaging, and Image Processing*, pages 854–859, September 2004.

[37] J.P. García-Ortiz, V.G. Ruiz, and I. García. Efficient Virtual Slide Telepathology Systems with JPEG2000. *Scientific Journal on Applied Information Technology*, 3:16–22, 2007.

[38] J.P. García-Ortiz, V.G. Ruiz, and I. García. Virtual Slide Telepathology Systems with JPEG2000. In *29th Annual International Conference of the IEEE EMBS (Engineering in Medicine and Biology Society)*, pages 880 – 883, Lyon, France, August 2007. IEEE Computer Society.

[39] J.P. García-Ortiz, V.G. Ruiz, I. Garcia, D. Müller, and G. Dimitoglou. An efficient prefetching strategy for remote browsing of JPEG 2000 image sequences. In *Image Analysis for Multimedia Interactive Services (WIAMIS), 2010 11th International Workshop on*, pages 1–4, 2010.

[40] J.P. García-Ortiz, V.G. Ruiz, M.F. López, and I. García. Interactive transmission of JPEG2000 images using Web proxy caching. *IEEE Transactions on Multimedia*, 10(4):629–636, June 2008.

[41] J.P. García-Ortiz, V.G. Ruiz, F. Marcano, and O.F. Roca. Stitching and remote browsing of images using JPEG2000 and JPIP. In *XV Winter Course of CATAI (Centro y Cursos de Alta Tecnología y Análisis de Imagen)*, pages 112–116, march 2007.

[42] International Organization for Standardization. Graphic technology - Prepress digital data exchange - Part 2: XYZ/sRGB encoded standard colour image data (XYZ/SCID). ISO/IEC 12640-2:2004.

[43] International Organization for Standardization. Information technology - Coding of audio-visual objects - Part 12: ISO base media file format. ISO/IEC 15444-12:2008.

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

[45] International Organization for Standardization. Information Technology - JPEG 2000 Image Coding System - Interactivity Tools, APIs and Protocols. ISO/IEC 15444-9:2005.

[46] International Organization for Standardization. Information Technology - JPEG 2000 Image Coding System: Extensions. ISO/IEC 15444-2:2004.

[47] International Organization for Standardization. Information technology - JPEG 2000 image coding system: Secure JPEG 2000. ISO/IEC 15444-8:2007.

[48] International Organization for Standardization. Information technology - Lossy/lossless coding of bi-level images. ISO/IEC 14492:2001.

[49] JPEG. Call for contributions for JPEG 2000 (JTC 1.29.14, 15444): Image coding system. Technical report, ISO/IEC JTC1/SC29/WG1, March 1997.

[50] K. Krishnan, M.W. Marcellin, A. Bilgin, and M.S. Nadar. Efficient transmission of compressed data for remote volume visualization. *IEEE Transactions on Medical Imaging*, 25(9):1189–1199, 2006.

[51] J. Li and H.H. Sun. On Interactive Browsing of Large Images. *IEEE Transactions on Multimedia*, 5(4):581–590, 2003.

[52] S.T. Liang and T.S. Chang. A bandwidth effective streaming of JPEG 2000 images using hypertext transfer protocol. In *IEEE International Conference on Multimedia and Expo*, pages 525–528, 2002.

[53] L. Lima, D. Taubman, and R. Leonardi. JPIP proxy server for remote browsing of JPEG2000 images. In *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*, pages 844–849, 2008.

[54] C. Lin and Y.F. Zheng. Fast Browsing of Large Scale Images Using Server Prefetching and Client Caching Techniques. In *Applications of Digital Image Processing XXII SPIE*, pages 376–387, 1999.

[55] H. Liu, X. Xie, W.Y. Ma, and H.J. Zhang. Automatic browsing of large pictures on mobile devices. In *ACM International Conference on Multimedia*, pages 148–155, November 2003.

[56] M.F. López, S.G. Rodríguez, J.P. García-Ortiz, J.M. Dana, V.G. Ruiz, and I. García. FSVC: A new fully scalable video codec. *Lecture Notes in Computer Science (LNCS). Proceedings of the 11th International Conference on Computer Analysis of Images and Patterns (CAIP2005)*, 3691:171–178, September 2005.

[57] M.F. López, S.G. Rodríguez, J.P. García-Ortiz, J.M. Dana, V.G. Ruiz, and I. García. Fully scalable video coding with packed stream. In *IS&T/SPIE Annual Symposium on Electronic Imaging Science and Technology*, pages 378–389, San José, California, USA, January 2005.

[58] M.F. López, S.G. Rodríguez, J.P. García-Ortiz, V.G. Ruiz, and I. García. Técnicas para la codificación escalable de vídeo. In *Actas de las XV Jornadas de Paralelismo. Computación de Altas Prestaciones*, pages 408–413, Almería, España, September 2004.

[59] A. Luotonen. *Web Proxy Servers*. Prentice-Hall, 1997.

[60] J.L. Monteagudo-Pereira, F. Auli-Llinas, J. Serra-Sagrista, and J. Bartrina-Rapesta. Smart JPIP Proxy Server with Prefetching Strategies. In *Data Compression Conference (DCC), 2010*, pages 99–108, 2010.

[61] D. Müller, B. Fleck, G. Dimitoglou, B.W. Caplins, D.E. Amadigwe, J.P. García-Ortiz, B. Wamsler, A. Alexanderian, V. K. Hughitt, and J. Ireland. JHelioviewer: Visualizing large sets of solar images Using JPEG 2000. *Computing in Science and Engineering*, 11(5):38–47, 2009.

[62] V.S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable Web server. In *USENIX 1999 Annual Technical Conference*, 1999.

[63] D. Pariag, T. Brecht, A. Harji, P. Buhr, A. Shukla, and D.R. Cheriton. Comparing the performance of web server architectures. *SIGOPS Oper. Syst. Rev.*, 41(3):231–243, 2007.

[64] W. Pennebaker and J. Mitchell. *JPEG still image data compression standard*. Van Nostrand Reinhold, 1993.

[65] W. Pesnell. The Solar Dynamics Observatory: Your eye on the Sun. In *37th COSPAR Scientific Assembly*, volume 37 of *COSPAR, Plenary Meeting*, pages 2412–+, 2008.

[66] E. Politou, G. Pavlidis, and C. Chamzas. JPEG2000 and dissemination of cultural heritage over the Internet. *IEEE Transactions on Image Processing*, 13(3):293–301, March 2004.

[67] A. Poulakidas, A. Srinivasan, O. Egecioglu, O. Ibarra, and T. Yang. Experimental Studies on a Compact Storage Scheme for Wavelet-Based Multiresolution Subregion Retrieval. In *Proceedings of NASA 1996 Combined Industry, Space and Earth Science Data Compression Workshop*, pages 61–70, 1996.

[68] R. Rosenbaum and H. Schumann. Grid-based interaction for effective image browsing on mobile devices. In *Proceedings of the SPIE*, volume 5684, pages 170–180, 2005.

[69] R. Rosenbaum and D. Taubman. Remote display of raster images using JPEG2000 and the rectangular FishEye-view. In *International Conference in Central Europe on Computer Graphics*, volume 11, pages 387–393, 2003.

[70] V.G. Ruiz, J.P. García-Ortiz, J. J. Fernández, J. M. Carazo, and I. García. Progressive image transmission in telemicroscopy. In *European Microscopy Congress*, pages 395–396, August 2004.

[71] J.J. Sánchez-Hernández, J.P. García-Ortiz, V. González-Ruiz, I. García, and D. Müller. Transmission of low-motion JPEG2000 image sequences using client-driven conditional replenishment. In *Proceedings of the International Conference on*

Signal Processing and Multimedia Applications, pages 11–16. SciTePress - Science and Technology Publications, 2011.

[72] M. Slattery and J. Mitchell. The Qx-coder. *IBM Journal of Research and Development*, 42(6):767–784, November 1998.

[73] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[74] N. Strobel, S.K. Mitra, and B.S. Manjunath. An Approach to Efficient Storage, Retrieval, and Browsing of Large Scale Image Databases. In *Photonics East, Proceedings of the SPIE —The International Society for Optical Engineering*, pages 324–335, 1995.

[75] Z. Su, T. Washizawa, J. Katto, and Y. Yasuda. Performance improvement of graceful image caching by using request frequency based prefetching algorithms. In *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology*, volume 1, pages 370–376, 2001.

[76] Z. Su, T. Washizawa, J. Katto, and Y. Yasuda. Hierarchical image caching in content distribution networks. In *Proceedings of IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, volume 2, pages 786–790, 2002.

[77] D.S. Taubman. High Perfomance Scalable Image Compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7):1158–1170, July 2000.

[78] D.S. Taubman. Remote Browsing of JPEG2000 Images. In *IEEE International Conference on Image Processing*, volume 1, pages 229–232, September 2002.

[79] D.S. Taubman and M.W. Marcellin. *JPEG 2000. Image compression fundamentals, standards and practice*. Kluwer Academic Publishers, 2002.

[80] D.S. Taubman and R. Prandolim. Architecture, Philosophy and Perfomance of JPIP: Internet Protocol Standard for JPEG2000. In *International Symposium on Visual Communications and Image Processing*, volume 5150, pages 649–663, July 2003.

[81] V. Tuominen and J. Isola. The application of JPEG 2000 in virtual microscopy. *Journal of Digital Imaging*, 22(3):250–258, 2007.

[82] C. Weidmann, M. Vetterli, A. Ortega, and F. Carignano. Soft caching: image caching in a rate-distortion framework. In *Image Processing, 1997. Proceedings., International Conference on*, volume 2, pages 696–699, 1997.