An efficient client-based JPEG2000 image transmission protocol

J.P. Ortiz, V.G. Ruiz and I. García
Computer Architecture and Electronics Dept.
University of Almeria, Spain

ABSTRACT

JPEG2000 is a new still image coding standard that allows to implement efficient remote image browsing applications. In this kind of developments, the clients retreives from a server only the desired regions of the remote images. In the Part 9 of this standard is defined the JPIP protocol, offering a complete set of syntaxes and methods for the remote interrogation of JPEG 2000 images. Using the JPIP protocol, the server side is where all the hard processes are done, so the clients have only to request the desired region and wait for the associated information. This little advantage has many other disadvantages like, for example, to not support the partial proxy caching, to difficult to implement improvement techniques like data prefetching or to offer a poor behaviour in wireless environments. A image transmission protocol with a more complex client side avoids all these disadvantages, only requiring a bit more process in clients. In this paper it is analyzed the current JPIP protocol, showing how affect its philosophy in the mentioned situations. A new JPEG2000 image transmission protocol is presented, J2KP, based on JPIP. Its performance is compared with JPIP and it is demostrated that, all the features rejected by JPIP offer a considerable increase of performance when are applied to this new protocol. Moreover, the process overload in the clients is nearly negligible.

Keywords: Remote browsing, JPEG2000, JPIP, web caching, prefetching.

1. INTRODUCTION

Nowadays, remote browsing of images is widely utilized in applications such as remote surveillance, electronic shopping, teleconferencing, medical diagnostic, remote astronomy, telemicroscopy and remote image databases, among others. For these applications, users visualize digital images stored in remote servers using the Internet as the communication media.

JPEG2000 is the most recent still image compression standard and its powerful features have turned it into one of the most frequently used for developing this kind of applications. This standard achieves high compression ratios; progressive visualization along the transmission time; and spatial scalability, allowing the users to retrieve the desired regions of the remote image. With this standard it is also possible to use either lossless or lossy compression.

The Part 9 of this standard defines a new protocol for the communication between servers and clients in these applications. One of its main advantages is the simplicity given to the client side, so almost all the intelligence, process load and communication control is passed to the server. This feature makes it ideal for mobile devices, with important resource limitations.

However, this simplicity in the client side affects negatively to the final performance of the protocol in different situations and communication architectures. These conditions are analyzed in this paper and it is proposed a new protocol, J2KP. This new protocol solves all the disadvantages of JPIP, only requiring a bit more process and intelligence load in the client side.

The paper is organized as follows: in Section 2 the JPEG2000 still image compression standard is briefly described and followed by Section 3 where the JPIP protocol is described. In Section 4 is detailed how to affect the server-based philosophy to the performance of JPIP. Our proposal is described in Section 5, and evaluated in Section 6. Finally, this article concludes with Section 7.

2. THE JPEG2000 IMAGE COMPRESSION SYSTEM

JPEG2000[5] is the last ISO/ITU-T standard for still image coding. Compared to JPEG, JPEG2000 obtains better compression ratios for the same image quality, specially at very low bit-rates. Some features of JPEG2000 are error-resilience, arbitrary shaped WOI coding, random access to the code-stream, multicomponent images, palletized color, binary images, compressed domain lossless flipping and simple rotation. These characteristics make JPEG2000 ideal for the coding and remote retrieving of large images.

JPEG2000 is based on the dyadic DWT (Discrete Wavelet Transform) that can be performed with either a reversible filter, which provides for lossless coding, or a non-reversible one, which is more suitable for higher compression ratios due to its linearity. Figure 1 shows how the DWT decomposes an image in $1 + 3r$ spatial frequency sub-bands, where r is the number of transform stages. Note that each band is composed of three sub-bands HL, LH and HH, where L stands for the application of the low-pass filter used in the DWT and H stands for the high-pass filter. The LL sub-band is always a low resolution version of the original image, the resolution level 0. The main advantages of the DWT domain are: (i) a low entropy level, (ii) a multiresolution representation, and (iii) a spatial-frequency representation of the image which enables a fine-grain distortion scalability when the wavelet coefficients are coded by bit-planes.

In JPEG2000 the DWT is followed by a quantization stage. It uses an embedded dead-zone scalar quantizer that in the reversible path is equivalent to select the bit-planes of the coefficients, starting at the most significant one. Then, each sub-band is divided into rectangular blocks (the so called *code-blocks*), typically of 64×64 or 32×32 coefficients, and progressively and independently coded using EBCOT (Embedded Block Coding with Optimized Truncation), an entropy codec based on binary arithmetic coding and context modeling.[2]

The code-blocks are grouped into rectangular blocks called *precincts*, whose dimension can change at each resolution level. The bit-streams of the code-blocks are divided into l segments, where l is the number of quality layers selected for the compressed image. The size of each segment depends on the contribution of the code-block to the quality layer.

The bit-streams of each code-block of a precinct, associated with the same quality layer, are grouped into a *packet*. An example of this bit-stream partition is shown in Figure 2. The JPEG2000 standard can also handle images that contain several components (multicomponent images) such as color RGB images. Each component is compressed independently, and therefore, each precinct only "speaks" about one of these components. Finally, in order to minimize the computational requirements of the codec or to increase the compression performance for some kind of images, JPEG2000 divides images into pieces called tiles. So, a packet is defined as the contribution

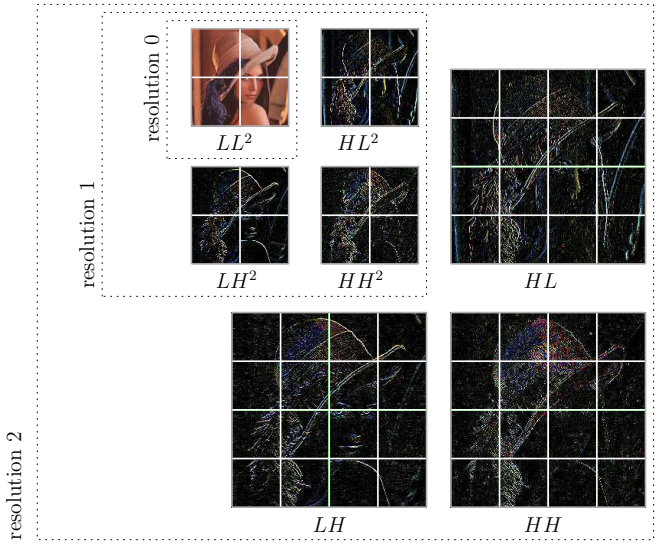

Figure 1. 2 levels DWT of *lena* (3 resolution levels), partitioned by precincts.

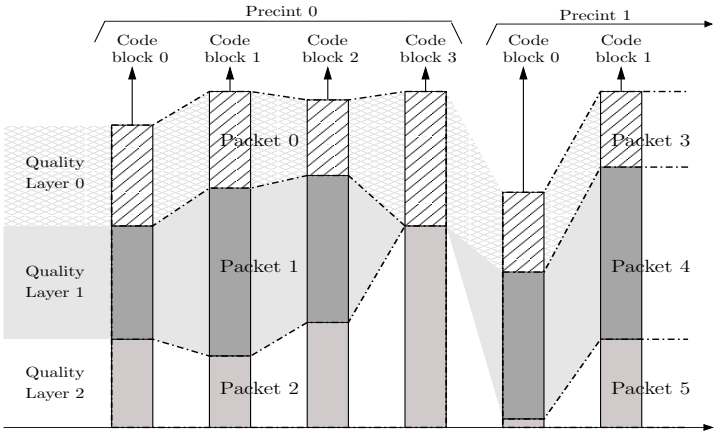

Figure 2. Example of the generation of packets.

(that could be zero, see Figure 2 for code-block 3 and quality layer 1) of a precinct to a quality layer of some tile, some component and some resolution level.

The JPEG2000 standard is described in several parts.[4,6] Part 1 (The Core System) describes how to build the simplest code-stream. A JPEG2000 code-stream contains the packets generated by the image compression process and a set of markers. Every marker consists of a 2-bytes word and can be used to reach one or both of the following goals: (i) to delimit a specific section of the code-stream, or (ii) to store useful information. Some markers are optional, some must appear in a specific order while others can be arbitrarily inserted within the code-stream.

At the beginning of the code-stream it is stored the main header. It consists of a set of markers which contain general information such as image size, number of components, etc. The main header is followed by the content associated with every image tile stream. Each image tile stream can be divided into one or more tile-parts, composed by a tile-part header and a set of packets. The tile-part header consists of a set of markers which contains specific information about the tile.

The order used to store the packets in every tile-part stream determines the default progression used by the decompressor to reconstruct the tile image. This ordering can be different for each tile of the image. The standard defines the following kinds of progressions: LRCP, RLCP, RPCL, PCRL and CPRL, where L stands for quality layer, R for resolution level, C for component and P for precinct.

3. THE JPEG2000 INTERACTIVE PROTOCOL (JPIP)

The JPEG2000 compression standard offers a set of features such as high scalability, efficient compression ratios and the possibility of random access to the code stream. This set of features makes JPEG2000 ideal for developing applications for the remote and interactive browsing of large images. However, this standard does not define any model for developing this kind of system and does not cover aspects like the server and client structures, the interaction between them or the protocol/s to be used.

To address this issue, S. Deshpande and W. Zeng[10] have proposed the use of the HTTP/1.1 protocol.[8] This solution offers very interesting features: (i) it does not require implementing a new specific protocol, (ii) it is possible to use the countless Web servers, as long as they support byte-ranging (for example the Apache Web server), and (iii) it exploits the caching system, although not very efficiently. Nevertheless, clients need to retrieve an extra index table that describes the content of the JPEG2000 file. This table (which is relatively large) must be downloaded before the request of any image data, significantly delaying the initial browsing of the image.

Other authors have devoted their efforts to improving the performance of the original solution.[3,9,11] Finally, the JPIK protocol,[1] designed by D. Taubman, was adopted by the JPEG committee with minor modifications. The final name of this protocol was JPIP (JPeg2000 Interactive Protocol)[12] and it can be found in Part 9 of the

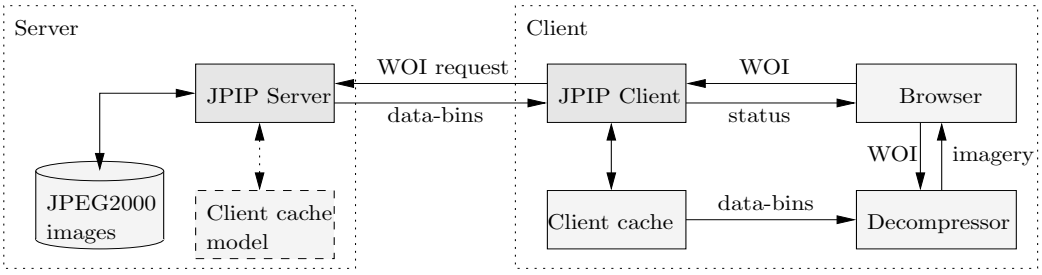

Figure 3. Client-server architecture for JPIP.

JPEG2000 standard.[7] JPIP covers the definition of a new protocol, as well as a framework to develop systems for remote browsing of JPEG2000 images. The JPIP protocol is the most efficient solution proposed until now.

Figure 3 shows a representation of the interaction scheme between client and server defined by JPIP. At the client side, the user specifies a WOI using the application browser. This WOI is passed to the JPIP client module and the decompressor module. The first one manages communications to the server, elaborating the corresponding request and reading the response from the server. When the information is read, it is stored in the internal cache. The decompressor module will gather the required cache information to reconstruct the WOI image, which will be shown to the user by the browser module.

All the JPEG2000 images accessible by the remote user are stored at the server side. When the JPIP server receives a client request for a specific WOI, it reads and processes the associated image extracting the appropriate information which must be sent to the client for reconstructing the requested WOI.

The server optionally maintains a model of the client cache, that is used to avoid resending redundant information to the client. In principle, the server assumes that all the information sent is cached at client side. Clients can modify its cache models maintained by the server adding some special parameters in their requests. For example, if a client needs to release some memory of its cache, it should notify the server of this, to keep the coherence of the cache model.

Both client cache and cache model, and the server responses are organized as structures named data-bins. Data-bins are blocks of data that JPIP uses to manage the JPEG2000 file contents. The data-bins have an initial header with information like, for example, the data length, identifier of the associated code-stream, etc.

There are several types of data-bins, for example, a data-bin that stores the main header information, another one intended to store the metadata contained in the JP2-family files, and another that contains all the packets associated to a precinct. The last one is perhaps one of the most important data-bin. The precinct data-bin stores all the packets of a precinct, sorted by quality layer, in increasing order. The data-bins can be sent divided in segments, allowing the server to interleave segments of differents data-bins for the transmission.

The JPIP standard defines two partitioning schemes for JPEG200 code-stream, based on either precincts (JPP) or tiles (JPT) as the predominant data-bins. For the JPP partitioning scheme, the image information is divided spatially into precincts and for the JPT partitioning scheme this division is made in tiles. In general the JPP stream will be used, because the tiling produces artifacts in the reconstructed image.

The JPIP protocol was designed to be independent of the base protocol used, only requiring to accommodate the structure of the messages to this one. The HTTP protocol is commonly used as a base protocol, so the client requests as well as the server responses are formed as canonical HTTP messages.

The server can send additional information, apart from data-bins, included by means of headers. If required, the server can modify any parameter requested by the client using these headers. The client does not always retreive exactly the requested information.

Using the HTTP as base protocol, two different ways of working exist: the simplest way is to use this protocol for both, the requests and the responses, as it is made in a common Web communication; the alternative is to use the HTTP protocol for the messages too, but using a secondary TCP for the data transmission. In this case, both the client and the server use HTTP messages, but the server responses do not contain data-bins; these are

sent through the TCP channel. The information sent through this secondary channel is segmented in "chunks", and the client confirms every received chunk to the server, allowing it to manage the data flow to maintain the network efficiency and responsiveness.

4. JPIP PERFORMANCE ANALYSIS

The JPIP protocol introduces interesting features for the development of applications for remote browsing. One of the most important is evidently the simplicity for the client, which becomes a mere intermediary between user and server. Implementation is, theorically, simple, and the process load which is necessary goes only during the decompression of the contents of the cache. The communication between client and server also becomes simplified, going almost all the control of it on the server. The client only has to transcribe the user interaction in a compressible format and send it to the server. In principle, a message for WOI or user interaction, almost without process, would be enough.

This simplification on the client side is not actually so important. The process load which would mean to give more intelligence to the client would not be comparable with that required to decompression and visualization of the code-stream. Any device able to perform this process skillfully would be also able to incorporate a higher complexity in the communication with the server.

In practice, the client does not use a single message for WOI. This would cause the impossibility of not being able to adjust to the variations of the band-width and a low efficiency of the communication in the changes in WOIs. The latter would happen, for example, if, after the server has sent the information associated with a required WOI, the client request it for another one. During the time this request takes to reach the server, the latter keeps on sending information of a WOI not longer required, and the client will also have to cache.

In general, the client specifies a maximum size of data to be sent by the server. The server stops the sending of information once the said maximum is reached. The client continually sends the same request, being able to modify the maximum, and the server, basing on the caching model, sends new increases in information. This behaviour, due to pipelining, has a performance similar to that previously discussed, but avoids its disadvantages.

The simplification on the client side proposed by the JPIP protocol affects notably different aspects of the performance of a final application, which next will be detailed:

- **Proxy caching support**

  The standard allows the server to modify freely the requests made by the client. Besides, the server could reorganize freely the information to be sent, but not the contents, either because of internal reasons or to adjust to the existing band-width. This freedom in the server performance makes that two similar requests seldom obtain the same response.

  As it was discussed in the Section 3, the JPIP is commonly implemented on HTTP, making that JPIP communications are realized without any problem through Web proxies (like Squid[18]), being treated like common HTTP communications. The standard dictates to include, in this case, the headers and parameters appropriate to avoid caching in the proxies, given the impredictible performance of the server.

  The Web caching system is globally used on the Internet to improve the efficiency of communications, removing redundant requests to the server, which are solved by intermediate proxies. There are plenty of proxies, most of them free, and many of them transparent. For example, many Internet service providers use transparent proxies to reduce the final traffic of the Web browsing of their clients.

  In a system for browsing of remote images, in which generally images have considerable sizes, the traffic between clients and server is very redundant. In other words, there is a high probability of the coincidence of several clients requesting the same or overlapped WOIs. The JPIP protocol does not allow to take advantage of this feature by means of the use of the existing Web proxies. The development and implantation of specific proxies would be necessary, and these should analyze the JPEG2000 flow, making the transcoding appropriate for the client.

- **Server scalability**

  Client-server architectures like the applications for remote browsing must be as scalable as possible, in other words, the number of simultaneous clients must not be a bottleneck in the global working. The architecture proposed by the JPIP limits more the scalability as much efficient it is implemented. The fact of simplifying the clients side causes a non scalable complexity on the server side.

  If the caching models are maintained for every client, there is an increase in the memory proportional to the number of active clients. These caching models must index and keep a mirror of the contents, data-bin to data-bin, of the client cache.

  When it comes to advanced servers, able to modify the clients requests according to the resources available or able to reorganize the content to be sent depending on the band-width available, there is another limitation in the scalability. This processing will be repeated for all the existing connections.

- **Prefetching support**

  Nowadays the remote browsing systems are provided with more and more intelligence, so that they learn from the final user performance, in order to give a smooth and suitable browsing. The interactions between the user and the images are not longer atomic: they are part of a registered path which allows to predict and anticipate the next moves of the user.

  For instance, A. Descampe et al.[15] propose a system in which the information to be sent by the server is ordered, not only taking into account the importance for the requested WOI, but also its relevance for the next WOIs estimated. Chengjiang Lin and Yuan F. Zheng[14] propose also a solution to anticipate the user behaviour according to its previous requests.

  Hao Liu et al.[16] proposed a solution to create an automatic browsing through large size images in mobile devices. It is a very attractive idea for these devices in which the user manoeuvring is limited. Though they do not make any reference to prefetching, there is no doubt that the performance of their novel solution would be increased significantly by means of this technique.

  If prefetching techniques are implemented on the server, it means another factor which limits its scalability. It would also means a lack of flexibility, given that the prefetching performance is bound to the browser GUI used by the user. For example, in the solutions previously discussed, an interaction based on zoom and panning is supposed, but there are another more advanced like that proposed by R. Ronsenbaum and H. Schumann,[17] also thought for mobile devices.

- **Transmission efficiency**

  Although it is possible to use different progressions for the sending of the JPEG2000 packets associated with a WOI, the most used one is LRCP. It offers the best performance in terms of PSNR x data received. As it was discussed in the Section 3, the packets are grouped in precinct data-bins and ordered within the same data-bin by quality layer. This means that, if the JPIP server wanted to used the LRCP progression for the transmission, it would have to divide the data-bins into so many segments as packets in it. This involves adding the additional load of the data-bin header in every packet sent. Depending on the granularity of the image JPEG2000 (code-block and precinct size, number of quality layers, etc.) and its size, this could be an important overload of information. To avoid this, the existing JPIP servers adjust dynamically the granularity depending on the estimated link bandwidth and the desired level of responsiveness.

- **Cache model coherence**

  The efficiency in the transmission is also directly connected to the coherence of the caching model that the server maintains for every client. The more exact is this model, the better will be the communication, avoiding the sending of redundant data. There are situations in which to maintain a good coherence of the caching model is rather difficult and it could entail an additional overload in the communication.

  When the client is running on terminals with heavy restrictions of memory, as the mobile devices, it will be necessary to release caching data-bins constantly. In this situation, if we wanted to keep a good coherence of the caching model in the server, the client should notify it all the changes going on. Depending on the

available memory, the frequency of these notifications may be important, involving an extra overload in communication.

If the JPIP protocol is used over communication channels prone to mistakes in the information or losses of packets in the communication, the keeping of the coherence becomes rather difficult. In some applications some errors and losses of packets are assumed to be, and when this happens retransmission is not allowed. However, in an common application for remote browsing, implemented on this kind of channels, it should not be blocked the retransmission of lost or incorrect elements. The problem is that, under the JPIP philosophy, a client cannot consistently determine if a packet is lacking in a given moment, because it does not request it explicitly. It is possible that the server decides to delay, even to cancel, the sending of the data-bin associated.

K. Munadi et al.[19] proposed an interesting solution to protect JPEG2000 streams against packet loss. Although it is oriented to applications without the possibility of retransmission, this work can be used to show how affects the packets loss to the transmission of JPEG2000 images.

5. THE J2KP PROTOCOL

The J2KP protocol (JPEG 2000 Protocol) inherits great part of the structure and syntaxes proposed by JPIP, analyzed in Section 3. In Figure 4 the proposed client-server can be observed.

The J2KP server must maintain constant the content of the images and its organization against clients, as long as these do not change. In this way, two identical requests must produce the same responses. This does not mean that the server must obligatorily keep the original organization of the image. For example, one organization or progression RPCL can be more efficient for storing the images, since it reduces the number of disk accesses. The server can maintain one different virtual organization facing the client. Therefore, it could store the images with a RPCL progression, but to offer to the clients a LRCP progression.

The possible caching must not be rejected nor disabled by the server. This last one should use the appropriate headers defined in HTTP/1.1[8] to control and to allow caching. With these headers, like for example `Cache-Control` or `Expires`, it can be defined, among other parameters, the time that the sent content will be kept constant by the server. During this time, it will not be necessary to refresh it. The caching can be made at client level as well as at intermediate Web proxys.

The content of the images stored at server side is grouped in contiguous blocks with a minimum size. For example, if 1 KB is defined as block size, this means that the blocks sent by the server contains the minimum number of consecutive packets with a total length of 1 KB or more. A numeric identifier or index is assigned to each block. The clients can requests all the necessary blocks to reconstruct a WOI, one by one, indicating their indexes.

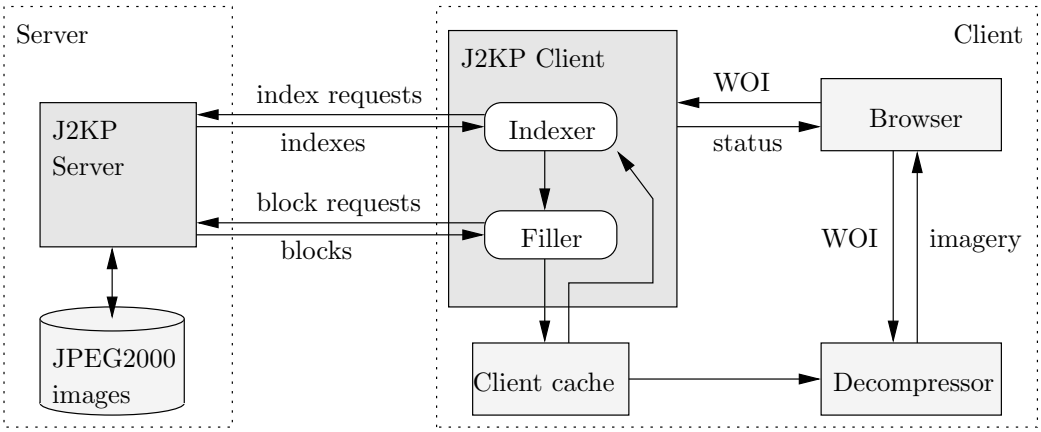

Figure 4. Client-server architecture for J2KP.

The requests made by the client are grouped in two different types: i) index requests , and ii) content or block requests. The index requests allow the client to obtain information about the necessary image blocks for a certain WOI. The server indexes the image and provides this information to the client. The block requests allow the client to retreive blocks, or ranges of them, from an image. The global functionality here proposed is similar to the one proposed by S. Deshpande and W. Zeng,[10] but in this case the client does not have to download any index file. The indexing process of the remote image is made through the J2KP server.

When the browser sends to the J2KP client the required WOI by the user, it determines which packets are necessary. As soon as the image header is read, the client can determine this information, as well as the image progression. After checking the existing content of the cache, the client requests to the server all the required index blocks for this WOI. This operation is carried out by the submodule *Indexer* represented in Figure 4. As the block indexes are received, they are passed to the submodule *Filler*. This submodule uses these indexes to request all the blocks through another communication channel. The received blocks are slitted in packets, considering the image progression, and stored in cache. It is checked that the received blocks contain the appropriate packets.

The index requests are made following the syntax defined by JPIP, that is, building an URL with a list of parameters that define the required WOI. An example of this kind of requests, using HTTP as base protocol, could be as follows:

```
GET /image.j2c?rsiz=640,480&roff=0,0&fsiz=1024,768&bsiz=1000&bexc=5-100 HTTP/1.1↩
Host:  get.j2kp.org↩
↩
```

The client must specify in the index requests the desired block size, using the `bsiz` parameter. This parameter, as well as the rest, can be modified by the server. Only the index requests can me modified by the server, as it occurs with JPIP. It is specially important that the server controls the block size used by the clients in order to keep coherent values in relation to the structure of the image. Notice that, to be able to modify this kind of requests does not mean that the server can vary the responses of the block requests.

The format used for this kind of requests makes that almost all the existing proxys reject caching the content of the responses. This format is typical for CGIs, with a very dynamic content and changeable in time. In any case, the responses of the index requests do not need to be cached. Their contents are usually short and there is a low probability for that two different clients make exactly the same request.

The server response is a list of indexes associated to all the necessary image blocks to generate the indicated WOI. With each index it is also necessary to send the useful ranges. Usually, the client does not need the whole content of the block, but only a range or ranges from it. In the implemented solution to the evaluation of this proposal it has been used a VBAS codification, defined in JPIP, in an incremental way, for the indexes as well as for the ranges. It could be possible to improve this codification, for example, compressing the content.

As the server does not maintain any client cache model, the index requests always produces responses with all the necessary block indexes. It can be a bit redundant during a normal client session. Using the `bexc` parameter the client can indicate all the blocks already stored in its cache, so the server does not include them in the response. In the previous example the client indicates that the blocks from 5 to 100 are already stored.

An example of a simple server response, for the previous request, could be as follows:

```
HTTP/1.1 200 OK↩
JPIP-bsiz:  4000↩
↩
<index><ranges>
<index><ranges>
...
```

In this example the server decides to modify the block size indicated by the user, giving it a value of 4 KB.

As the client receives the block indexes, it can request the associated blocks. This means that, as soon as the first block index is received, the client can send the first block request through the second communication channel. This does not involve any problem thanks to the pipelining.

The content or block requests do not have the same format that the one of the index requests, in order to be able to cache their responses. In this case, the blocks are referenced like a path file. For example, if the client requests the first 401 bytes from the block 3, it should send a message similar to the following one:

```
GET /image.j2c/bsiz/4000/blk/3 HTTP/1.1↩
Range:  bytes=0-400↩
Host:  get.j2kp.org↩
↩
```

Notice that the client must include the used block size. It is necessary to associate the block to the block size used for indexing the remote image. It allows the requests to be independents, making theirs responses easily cacheables.

The format used to specify the range of the block is the same as the defined one by the HTTP/1.1 protocol. The server must use the same format defined by this protocol for the responses. It is not obligatory to use it, and the client could decide always to retreive the whole block. Some proxys are configured for rejecting to cache the result of all the requests with ranges. Another proxys, however, depending of the object o block size, retreive and cache the whole content, but they send to the client only the indicated range. If the ranges are not used, the client usually obtain from the server redundant information, but probably useful for the next user movements.

As it can be observed, a bit of complexity has been added to the client. The server keeps certain control, but only for the indexing process of the remote images. With J2KP the client requests explicitly the necessary information. The philosophy of a dominant server is deleted, avoiding all its limitations.

6. EVALUATION

In principle, all the properties rejected or limited by the JPIP protocol are supported by J2KP. The server maintains invariable the remote images and their structures in the time with respect to the user requests. This makes possible the Web caching. Moreover, the information associated to a WOI is divided in blocks that are requested independently, exploiting more efficiently the proxy caching.

The philosophy proposed by J2KP does not show the same limitations of scalability that JPIP, not requiring neither processing nor additional memory by each client connection.

Being the client who explicitly requests the content of the WOI to show, has full control on the information that is transmitted. This facilitates the implementation of techniques for improving the browsing experience of the user, as the proposals cited in Section 4. This control also allows the client to detect and to correct possible lost of packets, when the protocol is implemented on channels prone to this situation, like the Wireless networks.

With J2KP the server does not need to maintain any cache model. Because of this, all the limitations imposed by this mechanism are not applicable. For example, in devices with few resources of memory, the continuous process of memory releasing/allocating, during a user session, neither generate additional traffic nor affects the performance.

The proposed protocol does not impose restrictions nor affects negatively to the use of the LRCP progression for the transmission of the image packets. This progression can be used as it without any extra load, neither of processing in the server, nor of information to send.

Probably, the most interesting feature offered by J2KP is the first one. To support the partial Web proxy caching can considerably increase the final performance of the application in many common situations. This property will be the only one analyzed and evaluated in this paper in detail. The rest of properties are not evaluated experimentally in this paper. The most of them, either are sufficiently documented in the indicated bibliography (like the prefetching techniques), or are very tie to the final implementation of the JPIP platform (like the scalability or the efficiency of transmission).

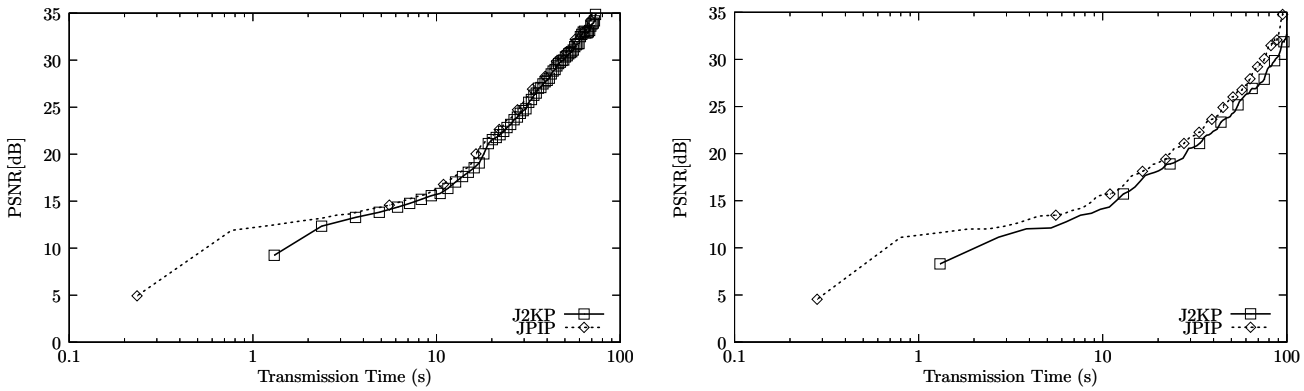

Figure 5. A comparison between J2KP and JPIP for two different WOIs for the same remote image.

For the experiments a client/server architecture has been used with a bandwidth for the communication limited to 4 KByte/s. The images stored in the server have been compressed using a progression LRCP and are segmented in blocks of a minimum block size of 4 KB. The client has a common GUI, that offers interactions based on zoom and panning. The browser behaviour is similar to many of the existing ones: when it is requested initially an image, this obtains a complete resolution of this one that fits in the screen of the user. Browser maintains by default the WOI dimensions similar to the view area of the application. It is assumed that the user has a standard dimension work of 1024×768.

The ranges have not been specified in the client block requests, in order to evaluate the worst case, generating redundant information for the server responses.

In a first experiment the performance of J2KP as opposed to JPIP has been evaluated. For this purpose different tests have been made with different images, considering the previous characteristics. In this section is devoted to describing a summary of the most important experimental results obtained from this set of tests. For the protocol JPIP the pair formed by the server `kdu_server` and the browser `kdu_show` were used, of the commercial package JPEG2000 Kakadu.[13]

In Figure 5 can be seen a graphic comparison between both protocols, at two different moments from a user session. The comparison is made in terms of PSNR versus time of transmission. In this test, the remote image has a size of 1970×2964. It makes that the JPIP client requests initially the complete image in a resolution level increased in 2 to fits in screen. Therefore, the initially requested WOI has a size of 492×741. Once downloaded this WOI, the user makes zoom, causing that the client requests another WOI of the same dimensions, but of a superior resolution level. In this test there is not any caching improvement.

Notice that the performance offered by the J2KP protocol is similar to the one of JPIP. Although this last one is more efficient, the difference is hardly significant. In the tests, JPIP responds quickly to the user interactions, being able to reconstruct one first reconstruction of the WOI faster than J2KP. Nevertheless, the quality of this first reconstruction does not offer a sufficiently legible image. The performance of the J2KP protocol becomes similar to the one of JPIP after a small initial time.

In a second experiment a proxy Squid is used between the communication of the server and two different clients, these two located in two different machines. The communication between the server and proxy has a bandwidth limited to 4 KByte/s, whereas the communication between proxy and the clients has a bandwidth of 10 MByte/s. One client makes several requests for the image previously used. All the content downloaded from the server is cached by the proxy. Next, the other client makes the same requests made in the previous test. In Figure 6 it can be seen the comparison of the new performance of J2KP as opposed to JPIP.

As it can be observed, the J2KP shows in this situation a performance considerably bigger. Considering the initial behaviour of the browser, when a client initiates the navigation through a remote image makes possible caching of a great amount of information in proxy. Due to this, if another user requires the same image, there will be a lot of available information in the proxy that could be downloaded quickly.

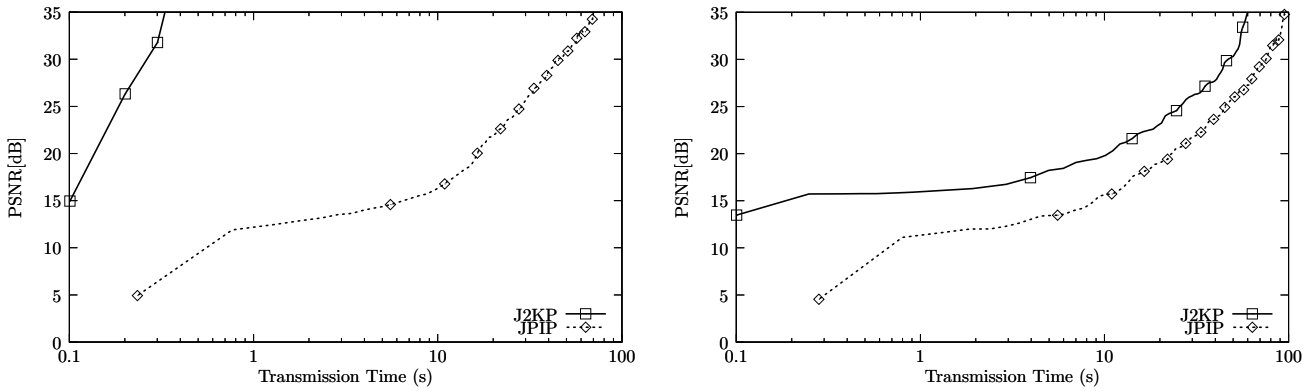

Figure 6. A comparison between J2KP and JPIP for two different WOIs for the same remote image (with caching).

7. CONCLUSIONS

J2KP, a new client-based protocol for the transmission of JPEG2000 images on the Internet, has been presented. The context where this work is situated is nowadays very common and many applications have recently been developed: a server, using the Web over the Internet, provides access to a set of large images with a low level of dynamism and that are required by an unknown number of users.

From the experimental results described in the previous section it can be concluded that the performance of J2KP is, although inferior, quite similar to the offered one by JPIP in a normal situation. However, when this protocol is used in Web environments with existing proxys, it is showed that its final performance is improved considerably. There are a lot of proxies and proxy hierarchies on the Internet; most of them available for their freedom of access and use. To be able to exploit them is a very powerful feature in remote browsing.

The J2KP protocol requires a bit more complexity than JPIP at client side. This additional complexity does not limit its use in mobile devices. Moreover, as it has been analyzed in this work, it accepts almost all the features either rejected or limited by this last one. Many of these features, like proxy caching support, can improve remarkably the performance of the applications.

REFERENCES

1. D. Taubman, "Remote browsing of JPEG2000 images", *International Conference on Image Processing*, pp. 229-232, 2002.
2. D. Taubman, "High performance scalable image compression with EBCOT", *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158-1170, Jul. 2000.
3. Shih Tsung Liang and Tain-Sao Chang, "A bandwidth effective streaming of JPEG2000 images using hypertext transfer protocol", *IEEE International Conference on Multimedia and Expo*, pp. 525-528, 2002.
4. ISO/IEC 15444, *Information Technology - JPEG2000 Image Coding System - Part 1: Core coding system*, 2000.
5. Majid Rabbani and Rajan Joshi, "An overview of the JPEG2000 still image compression standard", *Signal Processing: Image processing*, vol. 17, pp. 3-48, 2002
6. ISO/IEC 15444, *Information Technology - JPEG2000 Image Coding System - Part 2: Extensions*, 2000.
7. ISO/IEC 15444, *Information Technology - JPEG2000 Image Coding System - Part 9: Interactive tools, APIs and protocols*, 2003.
8. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol - HTTP/1.1*, http://www.ietf.org/rfc/rfc2616.txt, Jun. 1999.
9. J.P. Garcia-Ortiz, V.G. Ruiz, and I. Garcia, "Remote browsing of JPEG2000 images on the Web: evaluation of existing techniques and a new proposal", *IASTED International Conference on Visualization, Imaging, and Image Processing*, pp. 854-859, 2004.

10. Sachin Deshpande and Wenjun Zeng, "Scalable streaming of JPEG2000 images using hypertext transfer protocol", *ACM Multimedia,* pp. 372-381, 2001.
11. Jin Li and Hong-Hui Sun, "On interactive browsing of large images", *IEEE Transactions on Multimedia*, vol. 5, no. 4, pp. 581-590, Dec. 2003.
12. D. Taubman and R. Prandolini, "Architecture, Philosophy and Performance of JPIP: Internet Protocol Standard for JPEG2000", *Visual Communications and Image Processing*, pp. 791-805, 2003.
13. Kakadu, a comprehensive framework for JPEG2000, http://www.kakadusoftware.com/
14. Chengjiang Lin and Yuan F. Zheng, "Fast browsing of large scale images using server prefetching and client cache techniques", *Applications of Digital Image Processing XXII SPIE*, July 1999, pp. 376-387.
15. A. Descampe, Jihong Ou, P. Chevalier, and B. Macq, "Data prefetching for smooth navigation of large scale JPEG2000 images", *IEEE International Conference on Multimedia and Expo (ICME)*, July 2005.
16. Hao Liu, Xing Xie, Wei-Ying Ma, and Hong-Jiang Zhang, "Automatic browsing of large pictures on mobile devices", *Eleventh ACM international conference on Multimedia*, pp. 148-155, Nov. 2003
17. R. Rosenbaum and H. Schumann, "Grid-based interaction for effective image browsing on mobile devices", *Proceedings SPIE - Electronic Imaging: Multimedia on mobile devices*, Jan. 2005
18. Squid Web Proxy Cache, http://www.squid-cache.org/
19. K. Munadi, M. Kurosaki, K. Nishikawa, and H. Kiya, "Efficient packet loss protection for JPEG2000 images enabling backward compatibility with a standard decoder", *International Conference on Image Processing*, pp. 833-836, Oct. 2004