Improved JPIP protocol with proxy caching

J.P. Ortiz, V.G. Ruiz and I. García

Abstract—**This paper presents a set of minimal modifications of the JPIP protocol, with HTTP as transporting protocol, in order to exploit the proxy caching of the Internet infrastructure. Experimental results show important improvements in those Web applications designed for remote browsing of JPEG 2000 images using this solution.**

Keywords— **JPEG 2000, JPIP, HTTP, proxy, cache, data-bin.**

I. Introduction

JPEG 2000 is a recent image compression standard developed by the Joint Photographic Expert Group (JPEG) [1], [2]. It is based on the Discrete Wavelet Transform (DWT) and the Embedded Block Coding with Optimized Truncation (EBCOT). One of its main features is its high scalability; this scalability can be obtained in three different domains: quality, resolution and spatial.

The basic format of a JPEG 2000 image consists of a set of markers which contain references to useful information (compression parameters, image features, beginning or end of a packet, etc.), and also to a set of variable length packets, each of them containing the compression result of a specific area of the original image.

The order in which packets are stored determines the kind of progression when the image is decompressed sequentially. This order is described by a sequence of four characters which indicates the kind of scalability used (L= quality layer, R = resolution, P = precinct, C = component). For example, the RLCP order indicates that packets are stored by resolution (R), by quality layer (L), by component (C) and by precinct (P).

This progression establishes the default progression existing in the code-stream, but it is possible to access to the desired packets independently, and to retrieve them in any order.

In order to offer a higher flexibility and functionality to image files, JPEG 2000 defines a highly configurable and extending file format. This file format is based on "boxes" and permits to encapsulate one or more code-streams and to include diverse additional information.

The high scalability offered by the JPEG 2000 standard makes it ideal for applications for progressive and interactive remote images browsing of images. Because of the user's interaction, in these applications, a client carries out requests to a server asking for a specific region within a specific resolution level. The client also has the ability to indicate others parameters like the number of quality layers or the number of desired components.

Part 9 of the JPEG 2000 standard proposes a server/client architecture and a communication protocol called JPIP [3], [4] for the implementation of this kind of progressive and interactive applications. In this architecture, depicted in Figure 1, the client indicates to the server a region and the highest resolution level within a remote image (WOI: Window Of Interest). The JPIP server would send to the client the necessary information to reconstruct the requested WOI. This information is collected by the server from the local JPEG 2000 image.

JPIP protocol can be implemented over different protocols, being HTTP/1.1 protocol [5] one of the most interesting, since it easily allows to encapsulate requests and responses in HTTP messages, The use of HTTP/1.1 protocol permits to exploit the Web infrastructure.

The Web infrastructure offers many advantages with relation to another more specific, as for example, the caching system, which can be used not only at a client level, but also at the level of the existing proxies in the connections between clients and servers.

The JPIP protocol, running on HTTP, does not exploit all the performance of the Web caching system. The system proposed in this paper tries to exploit as much as possible the proxies caches in applications for progressive and interactive remote browsing of JPEG 2000 images.

The rest of this paper is structured as follows: in Section 2, the current working JPIP architecture is detailed, showing its deficiency for the task of exploiting the Web caching system, and proposing a solution for it; in Section 3, a possible implementation of the suggested system is explained; in Section 4, the achieved results with the proposed system are exposed, and finally, in Section 5, we finish with the pertinent conclusions about our proposal.

II. The proposed system

The JPIP protocol divides any JPEG 2000 image I (either in raw format or in a more complex format) into a set of N parts called data-bins, $I = \{d_1, d_2, \ldots, d_N\}$, so that, when a client at an instant t requests a specific window of interest of the image I, $WOI_t^{(I)}$, the information that the server sends in response, $r(WOI_t^{(I)})$, is the subset of image data-bins that permits to reconstruct that request, $F(WOI_t^{(I)})$. Therefore,

$$r(WOI_t^{(I)}) = F(WOI_t^{(I)}) = \{d_l, \ldots, d_k\} \subset I \quad (1)$$

As it can be observed in Figure 1, the server can optionally maintain a client cache model in order to

This work was supported by the MCyT under contracts TIC2002-00228.

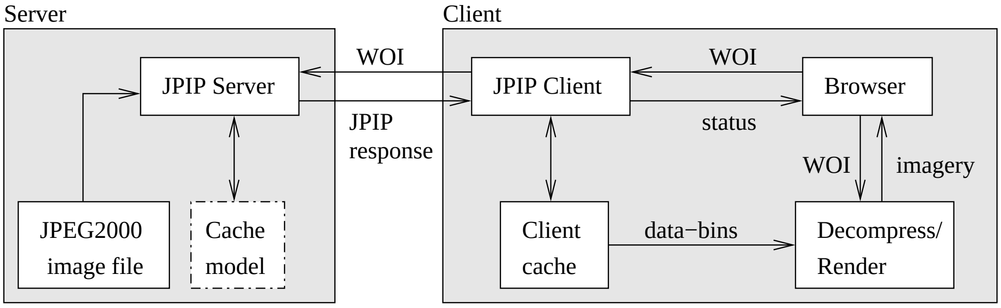

Fig. 1. Client-Server JPIP architecture.

avoid sending information (data-bins) that it has already sent. It is only incumbent upon the requests of a same client.

In Table I, the different types of existing data-bins are enumerated. It has to be pointed out that the most important data-bins are those concerning to the code-stream packets (Precinct data-bin).

The Web infrastructure caching system that we are going to use is maintained by the existing proxies in the connection between client and server. A proxy is a special server that acts as intermediary between one or more clients and one or more servers (both clients and servers can also be themselves proxies), in order to offer, among others, one or more of the following functions:

- Protocol conversion: In case clients and servers use different protocols.
- Security control: Allowing to establish a set of security rules in connections, in one or other direction.
- Caching system: Allowing to reduce the latency in case two or more clients ask for the same request in a sufficiently short interval of time.

With regard to the JPIP protocol, it is true that a proxy stores in its cache the results of previous requests, however a client can only take advantage of the information saved at the cache when its request is identical to a previous one.

This is rather inefficient, since many of the different requests for a same image will share information. Currently, for two different requests from the same image, from two different clients, that require a common set of data-bins, the server response for the last request is complete, as it appears in Equation (2), although had existed a proxy/cache hierarchy in the communication between client and server. More concisely,

$$\begin{gathered} P(WOI^{(I)}_{t+\Delta t}) \neq \emptyset \Rightarrow \\ r(WOI^{(I)}_{t+\Delta t}) = F(WOI^{(I)}_{t+\Delta t}) \end{gathered} \tag{2}$$

where $P(WOI^{(I)}_{t+\Delta t}) = F(WOI^{(I)}_{t}) \cap F(WOI^{(I)}_{t+\Delta t})$ is the subset of the necessary data-bins for the reconstruction of $WOI^{(I)}_{t+\Delta t}$ that were also used for $WOI^{(I)}_{t}$.

The most efficient behavior is when $P(WOI^{(I)}_{t+\Delta t})$ is saved into the proxies, and the server only has to send the rest of necessary data-bins that are not saved. This is given by:

$$r(WOI^{(I)}_{t+\Delta t}) = F(WOI^{(I)}_{t+\Delta t}) - P(WOI^{(I)}_{t+\Delta t}) \tag{3}$$

It is true that, due to the features of the standard JPEG 2000, it is possible to implement an application for remote image browsing, without necessarily using the JPIP protocol. For example, it would be possible only to use the HTTP/1.1 protocol and, using byte-ranging, a client would access to those necessary data-bins for a certain WOI. The problem is that most of the proxies do not manage byte-ranges efficiently, because byte-ranges are either ignored or they produce the proxies to retrieve from the server the whole associated file and, after received, they sent the client the requested byte-ranges. If byte-ranges are ignored then they pass through proxies without caching. So, it can not be assured that the possible existing proxies between a client and the final server make an efficient caching of the byte-ranges.

The first solution consists of dividing a single WOI request into a request of data-bin references, and a set of requests, one for each data-bin reference. Thus, the client would send to the server the WOI that it needs to browse, and the server would return the necessary data-bin references. The client would ask the server for one by one every data-bin, in different requests. This would make that, when encapsulating every request in HTTP message, proxies could cache them in an independent way, making Equation (3) possible. It would only have to add to the JPIP server the possibility of answering the requests of (i) the references of the necessary data-bins for a WOI and (ii) a specific data-bin. Besides, the JPIP server had to implement the HTTP protocol caching system, handling the different associated headers (Cache-Control, If-Modified-Since, etc.).

A priori, this first solution would efficiently use the caching system of the Web infrastructure, besides it would not need the caching system of the protocol JPIP. However, this solution has a overload problem because of the large amount of the own ASCII headers of the HTTP. Depending on the type of the messages, an HTTP message can have 200 additional bytes on average, due to the ASCII headers. To reconstruct an area of a JPEG 2000 image of

TABLE I

Data-bin types.

Type	Information
Precinct data-bin	Precinct data
Tile-header data-bin	All tile-part headers concatenated of a tile
Tile data-bin	All tile-parts concatenated of a tile
Main-header data-bin	Main header
Metadata-bin	Collection of boxes of a JPEG 2000 family file

2954×1976, with 7 resolution levels and 10 quality layers, with a precinct size of 128×128, we need a total of 1260 packets, for what we would have 252000 overload bytes on average when effecting the corresponding data-bins requests.

Besides, this overload per request is not homogeneous, so that for very large data-bins, the overload is small, but for very small data-bins (for example, the data-bin associated to an empty packet, of a 1 byte), the overload can be larger than 20000%.

To find a solution to this problem, we propose to create a set of blocks, each containing a subset of data-bins. For an image I, we define a minimum block size $s^{(I)}$, and we join the data-bins of the image into blocks, so that each block contains the minimum number of data-bins such that the total size of each block is equal or greater than $s^{(I)}$. Notice that we only can assure this for all blocks except the last one.

For a minimum block size equals to 0, we would find that a block would be identical to a data-bin. The minimum block size would be chosen depending on the average size of the image data-bins.

When making a request per block, instead of per data-bin, we control the overload produced by the HTTP headers. For any response to a request of any block, we would have a maximum overload of $(100 \cdot h/s^{(I)})\%$, where h is the maximum size of the block headers. This overload assumes that, when requesting a specific block, all blocks included are necessary, although many times this is not true. In any case, it would comply with (3) at the block level, with a time delay reduction in responses.

The most important data-bin is the one associated to the packets, so if we would want the blocks to include the maximum number of interesting data-bins, it would be recommendable that the last progression dimension was by precinct, P (for example, RLCP), that is, the packets would be organized by lines, within the code-stream.

III. Implementation

The implementation of this proposed solution is not intended to replace the JPIP's current, but for being a complement to use in those applications in which it is required to use efficiently the caching system of the Web infrastructure. Next we propose the basic modifications to do to a classic JPIP server in order to implement the proposed solution.

Firstly, we have to modify the server for accepting a new type of request, with which a client can ask for the references of the necessary blocks to reconstruct a specific WOI. For this, we will include the parameter `request`, with the value `blocks`, in a request. So, for example, a request of a client asking for the references of the necessary blocks to reconstruct a WOI with a size of 310×310, with an origin $(10, 10)$, and at a resolution level within an area of 512×512, for the image *rimage.jp2*, would be as follows:

```
GET http://jp2.server/rimage.jp2?roff=
10,10&rsiz=310,310&fsiz=512,512&request=
blocks HTTP/1.1↩
Host: jp2.client↩
↩
```

When the server finds the `request` parameter with the `blocks` value, it provides the references of the necessary blocks to reconstruct the required WOI. The client can also indicate the wanted minimum block size, in bytes, with the `sblock` parameter. This size could be either accepted or not by the JPIP server, being the server able to modify it, if necessary, notifying the client this modification by means of `JPIP-sblock` header. If the client does not specify any block size, the server will use a default size and will notify the client.

Next we can see an example, with the same last request, but specifying 512 bytes as the minimum block size required:

```
GET http://jp2.server/rimage.jp2?roff=
10,10&rsiz=310,310&fsiz=512,512&request=
blocks&sblock=512 HTTP/1.1↩
Host: jp2.client↩
↩
```

The server could provide a response as follows, where the server has changed the minimum block size to 256 bytes:

```
HTTP/1.1 200 OK, with modifications↩
JPIP-sblock: 256↩
↩
...
```

If the server did not find the `request` parameter, its behavior would be the standard one. If it was found, it must not include headers for disabling proxies caching, like `Cache-Control: no-cache`, but it must be able to interact with the Web caching system, supporting headers like `If-modified-since`,

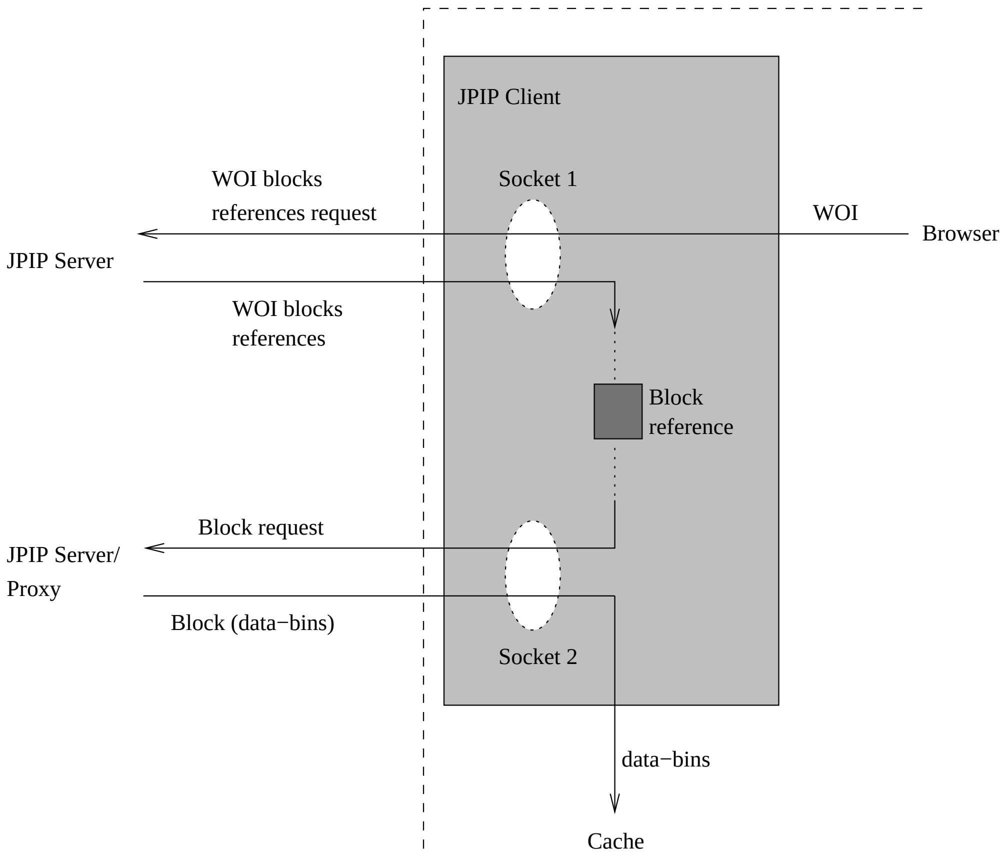

Fig. 2. JPIP client proposed.

and ignoring the headers of the caching system of the JPIP protocol.

The content of a server response is a set of references of the necessary blocks to reconstruct the requested WOI. Every block reference is an index, starting at 0, with a VBAS structure (variable-length byte-aligned segment), defined in JPIP. This structure, which can be observed in Figure 3, allows to store a number B, whose binary representation would have a length of L_B bits, in a total of $\lceil L_B/7 \rceil$ bytes.

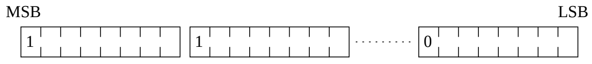

Fig. 3. VBAS structure.

This first server response will not be cached, due to the fact that two identical requests can produce different server responses (a server could modify any request parameter). Because of this, it would be more interesting that the server response was as small as possible, performing the compression. It is carried out by the HTTP protocol using the `Content-enconding` header. With this header, the server could compress the response content, indicating which algorithm, within the set of supported algorithms of the protocol, has been used (for example, it could indicate `deflate`, `gzip`, etc.).

In Figure 2 we can see the JPIP client structure, with the proposed modification. This structure needs two sockets, Socket 1 and Socket 2. It would send references requests through Socket 1 and, when it is received any reference, it would send the associated block request through the Socket 2. In the proposed client structure, the own pipelining of the HTTP/1.1 is fully exploited, allowing to make requests and to receive responses both in parallel.

As soon as the client receives a block reference through the Socket 1, it sends a request about the referenced block through the Socket 2. In order to request a specific block, it is necessary to use the `block` parameter, taking the value of the wanted block index. In the following example we can observe a client request about the block number 20, of a remote JPEG 2000 image called *rimage.jp2*:

```
GET http://jp2.server/rimage.jp2?block=
20&sblock=256 HTTP/1.1↩
Host: jp2.client↩
↩
```

For all the requests of the client, it is necessary it to indicate the minimum block size used, which will be used by the server to join the image data-bins in blocks, and indexing them. This size must be the same than the indicated by the server as result of the first client request about the references of the blocks.

Every read block contains a set of contiguous data-bins. In order to offer the maximal compatibility, the format of these data-bins is the same that the format employed in the JPIP protocol. The order in which the server merges data-bins in blocksis completely free, although it would be more efficient to use an order that would allow to improve the progressive visualization in the client. In the same way, the order of the references sent by the server, due to a WOI request, is free too. However, for a simple format of a JPEG 2000 code-stream, the best ordering is that where the block containing the main header data-bin was firstly sent (and it would be recommendable that its position within the block was the first one); it would be also recommended to use an ordering in which the packet data-bins follow a quality progression, like, for example, LRCP.

The server must comply with that, having the same minimum block size and the same block number, two different requests would produce the same set of data-bins. This must be complied strictly to avoid incoherence in the cache of the proxies.

IV. Results

For the realization of the tests, it has been employed the architecture shown in Figure 4. In this architecture it can be seen that the connection between the clients and the server is carried out through one (or more) Web proxies, with a caching system. Several WOIs, all referring to the same image, are visualized in different moments of time, in different clients. The JPEG 2000 remote image used here is 2954 × 1976, with 7 resolution levels and 10 quality layers. In order to simplify the experimental test, the image format will be the raw format (*.j2c*) and it will not contain neither tile nor tile-parts.

The LRCP image progression has been used. The data-bins order was as follows: the first data-bin is the main header data-bin, and the next data-bins are the packet data-bins in the same order as the image progression.

As minimum block size 512 bytes was chosen, and having an average header size of 100 bytes per HTTP message, a maximum overload of 20% was obtained.

For recording the experimental results, a specific WOI was first visualized in one of the clients at the maximal resolution level. This WOI had a size of 512, its origin at $(100, 100)$. After this visualization, another different client will visualize a WOI with a size of 512, with $(356, 356)$ as origin, and at the same resolution level. These two visualizations were made for each system to compare the classic JPIP system to the proposed system.

In Figure 5 the values of the PSNR (in dB) of the WOI obtained at every time t for the first 50 seconds of the communication are shown. Here, it is supposed that the server connection bandwidth is on average 4KB/secs. On the other hand, the connection bandwidth of the first proxy in the client/server connection is only limited by the architecture of the local net where the first proxy is situated. For the experimental tests, it has been supposed that this bandwidth is 4MB/secs.

Experimental results in Figure 5 show that the speed of the reconstruction of the WOI, for the second visualization in the proposed system, is quite higher than the classic JPIP system. It means that, at any time, the Cached JPIP system obtains better quality of the image than the classic JPIP system. This speed would be incremented as much as more requests of the same image are made.

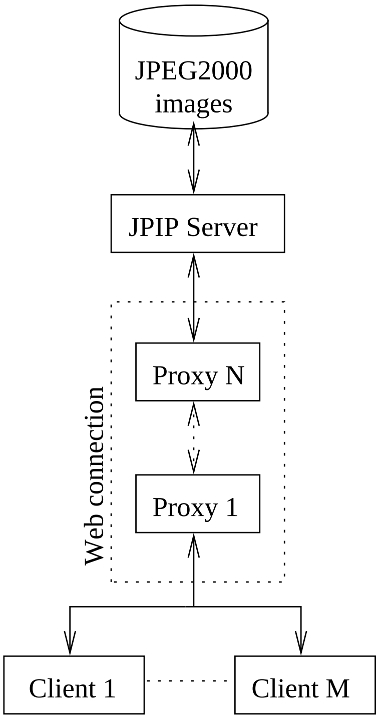

Fig. 4. Architecture used in the experimental tests.

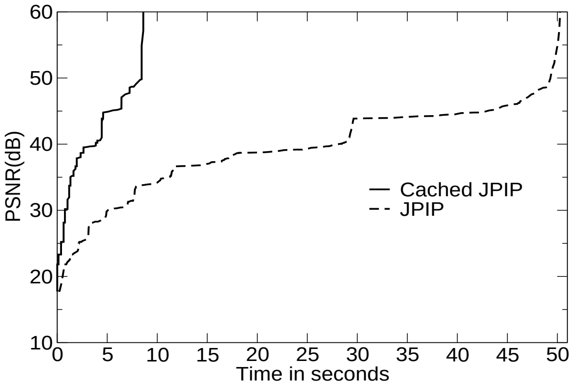

Fig. 5. Rate-distortion results from experimental tests.

In this simulation tests, the delay produced by the reception of the blocks references was not taken into account. It was done in this way because of the following two reasons:

- the requests of blocks are carried out in parallel with the reception of the block references,
- it is possible to compress the first server response, exploiting the feature of the algorithms proposed in the HTTP/1.1 protocol that allows to decompress the data “on-the-fly”.

V. Conclusions

In image transmission systems similar to that depicted in Figure 4, frequently appearing in the Internet, our experimental tests have shown that the classic JPIP system is rather inefficient. It has been proved that this happens because JPIP does not exploit the redundancy existing in the requests of different clients for WOIs of a set of remote images. This redundancy can be absorbed quite efficiently by the Web proxies using the proposed system (Cached JPIP). This proposed system coexists with the classical JPIP system and it requires minimal modifications.

References

[1] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 Still Image Compression Standard," *IEEE Signal Processing Magazine*, pp. 36–58, September 2001.

[2] ISO/IEC 15444, *Information Technology – JPEG2000 Image Coding System – Part 1: Core coding system*, 2000.

[3] ISO/IEC 15444, *Information Technology – JPEG2000 Image Coding System – Part 9: Interactive tools, APIs and protocols*, 2003.

[4] D.S. Taubman and Marcellin. M.W., *JPEG2000. Image Compression Fundamentals, Standards and Practice*, Kluwer Academic Publishers, 2002.

[5] 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, June 1999.