JPIP Proxy Server for remote browsing of JPEG2000 images

Livio Lima [#1], David Taubman [*], Riccardo Leonardi [#2]

[#] *Department of Electronics for Automation, University of Brescia*
Via Branze, Brescia, Italy
[1] livio.lima@ing.unibs.it
[2] riccardo.leonardi@ing.unibs.it

[*] *School of Electrical Engineering and Telecommunications, University of New South Wales*
Sydney, NSW, Australia
d.taubman@unsw.edu.au

***Abstract*—The JPEG2000 image compression standard offers scalability features in support of remote browsing applications. In particular Part 9 of the JPEG2000 standard defines a protocol called JPIP for interactivity with JPEG2000 code-streams and files. In client-server application based on JPIP, a client does not directly interact with the compressed file, but formulates requests using a simple syntax which identifies the current "focus window". In this kind of application particularly useful could be a proxy server, that potentially can improve the performance of the system through a better use of the network infrastructure. The aim of this work is to propose a proxy server with JPIP capabilities and shows the benefits that can be brought to remote browsing applications.**

I. Introduction

The JPEG2000 image compression standard offers scalability features in support of remote browsing applications, in addition to better compression performance compared to the previous standards in image compression. Some useful features are resolution scalability, progressive refinement and spatial random access.

The first approach proposed for the remote browsing of JPEG2000 images was the use of byte range access capabilities offered by the HTTP/1.1 protocol [1]. In this case, the client reads an index table to determine the locations of the relevant compressed data and header and explicitly insert the byte ranges required in the HTTP requests. Even if this approach requires a very simple architecture, the main drawbacks are the low level of flexibility and low performance because there is no possibility for the server to stream data to the client in a rate-distortion optimized manner.

In order to fully exploit the potential of the JPEG2000 standard for interactive applications, the ISO/IEC Joint Technical Committee of Photographic Experts (JPEG) defined Part 9 of the JPEG2000 standard, for interactivity with JPEG2000 code-streams and files. This part of the standard defines a protocol called JPIP [2][3]. In JPIP, a client does not directly interact with the compressed file, but formulates requests using a simple syntax which identifies the current "focus window", that defines the region of interest, the resolution and the components of interest for the client application. In reply to a JPIP request, the server is free to determine the most appropriate response elements in order to optimize the image quality available at the client.

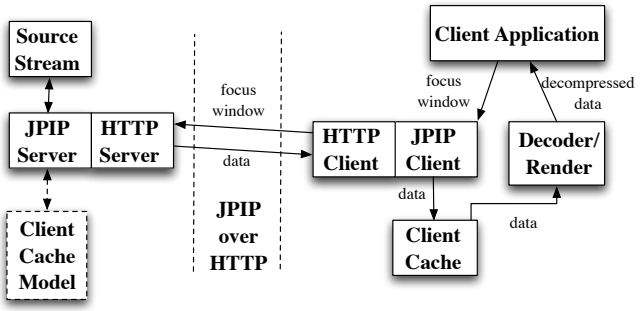

Fig. 1. JPIP client-server architecture.

A simple client-server architecture based on the JPIP protocol is shown in Figure 1. In real applications multiple clients are typically connected to the same server; moreover, the network infrastructure is more complex, including other network devices like proxies. In particular the use of a *proxy server* can improve the performance of the whole system through a better use of the network infrastructure. The main problem is that, at the moment, proxy servers used in this kind of architecture are only HTTP proxies, without any JPIP capabilities. So, for example, if a client makes a request of a focus window that is contained within the focus window previously requested by another client, the HTTP Proxy Server considers these two requests to be completely different, without understand that it is possible to reuse the data already received from the server in reply to the first request also to serve the second request. In this scenario the network link between proxy and *origin server* is not efficiently used because the same content is request multiple times.

In order to fully exploit the potential of the JPIP protocol and the JPEG2000 standard with an efficient use of the network infrastructure, the proxy should understand the JPIP

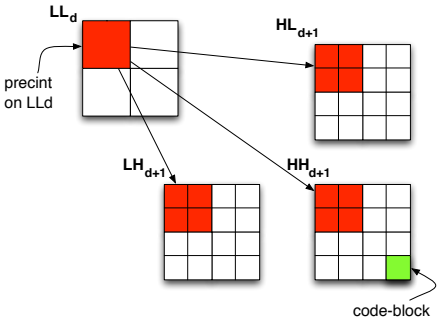

Fig. 2. Relationship between precincts, resolutions and code-blocks.

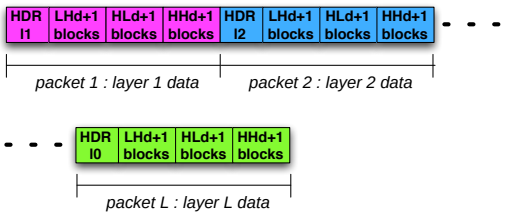

Fig. 3. Data stream for a single precinct on resolution LL_d.

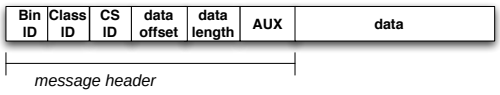

Fig. 4. Structure of a JPIP message.

protocol and use efficient algorithms to manage the JPIP requests and replies. This work concerns the development of a *JPIP proxy* and shows the benefits that can be brought to remote browsing applications.

Section II briefly reviews the organization of the JPEG2000 code-stream, while section III gives an overview of the JPIP protocol. In Section IV the description of the proposed JPIP proxy is given, while Section V presents experimental results.

II. JPEG2000 ELEMENTS

JPEG2000 is based on the Discrete Wavelet Transform (DWT), together with Embedded Block Coding with Optimized Truncation (EBCOT). D stages of DWT analysis decompose the image into $3D$+1 subbands, labeled LH_d, HL_d, HH_d and LL_D, for $d = 1, ..., D$.

Each subband is partitioned into rectangular blocks called code-blocks, each of which is independently coded. Resolution scalability is obtained by discarding the code-blocks of detail subbands and omitting the final DWT synthesis stage. Quality scalability is obtained through a "quality layers" abstraction. Each layer represents an incremental contribution (possibly empty) from the embedded bit-stream associated with each code-block in the image. Discarding one or more layers (starting from the highest one) produces a representation of the code-block with lower quality. Spatial random access is possible because each code-block is associated with a limited spatial region, typically 32×32 or 64×64 pixels, and independently coded.

Given any spatial region of interest within a particular resolution, it is possible to determine the set of code-blocks which contribute to the reconstruction of that region. JPEG2000 also defines collections of spatially adjacent code-blocks as "precincts." Figure 2 illustrates the relationship between precincts, resolutions and code-blocks.

Each precinct of resolution level LL_d consist of the code-blocks corresponding to the same spatial region within the subbands LH_{d+1}, HL_{d+1} and HH_{d+1} if $d < D$, or within the subband LL_D if $d = D$. The data-stream associated with each precinct is organized as a collection of "packets," one for each quality layer, as shown in Figure 3.

III. JPIP OVERVIEW

JPIP defines a protocol consisting of a series of interactions between a client and a server to enable request and transfer of selected content from JPEG2000 family files, codestreams and other elements. Basically, a client uses the JPIP syntax to define a request that includes a "Focus Window." The Focus Window specifies the interest of the client in terms of resolution, size, location, components, layers, and other parameters for the image. The server replies by delivering the image data, using a precinct-based stream, a tile-based stream, or else the whole image.

For the purpose of exchange and storage of JPEG2000 compressed data, JPIP defines a means of partitioning the JPEG2000 codestream into a collection of so-called "data-bins." JPIP support two partitioning schemes, one based on precincts and the other on tiles. In this paper we consider only the more flexible case of precinct data-bins. A precinct data-bin represents the sequence of JPEG2000 packets which contain coded information for the precinct. A single JPIP message contains any contiguous subset (byte range) from a single data-bin. Each message contains a header, identifying its data-bin and byte range, while a concatenated sequence of messages makes a stream. The structure of a message is given in Figure 4. For the purpose of understanding the policies applied by our proposed JPIP proxy, the *data offset*, *data length* and *AUX* fields need clarification. *Data offset* is the location within the precinct's data-bin, from which the first byte in the data field is taken; *data length* is the number of bytes in the data field. The *AUX* is an optional field, which the server may use to provide information about the quality layers represented by the message. Specifically, the *AUX* field identifies the number of complete quality layers which a client would receive if the byte range represented by the message were extended to include the start of the data-bin.

In order to take advantage of the *AUX* field information, the client has to explicitly request this functionality through the "EXTENDED HEADER" capability syntax element of the JPIP protocol. Since usually the server is free to choose the

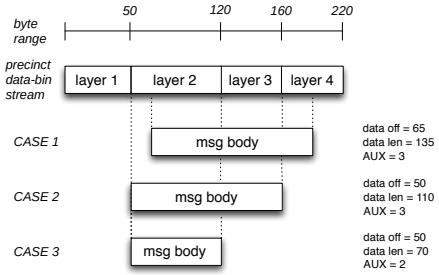

Fig. 5. Examples of JPIP messages.

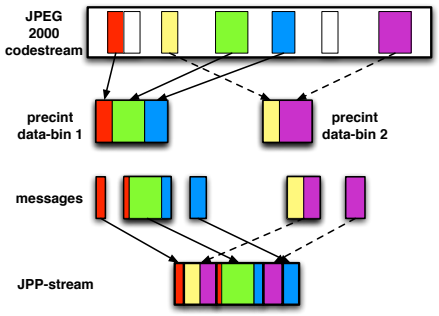

Fig. 6. Relation between JPEG2000 codestream, data-bins, messages and a JPIP stream.

data to send to the client, different situations may arise, as shown in Figure 5. In Case 1, a single JPIP message includes parts of 3 different quality layers. In this case, the client can deduce that layer 3 is fully contained within the message, but it cannot deduce the boundaries of any of the layers. In order to discover layer boundaries, the client may use the "ALIGN" JPIP syntax element in the request; this forces the server to include a whole number of packets (layers) within each message, as illustrated for Cases 2 and 3 in Figure 5. Even with the align option, the server may include multiple layers in each message, as in Case 2. This often improves communication efficiency, but it means that the proxy server cannot always determine the exact composition of each precinct's data bin in terms of layers.

The relation between JPEG2000 codestream, data-bins, messages and an entire JPIP stream is shown in Figure 6.

IV. JPIP Proxy Server

In the following sections, we use the terms *proxy* and *server* to refer to the Proxy Server and Origin Server[1]. In computer networks, proxies are placed at specific key points between the user and the server. Proxies service the requests presented by their clients, by forwarding these requests to servers or other proxies, or by using the responses cached from previous requests for the same resource. In remote browsing applications, a client connects to the proxy, requesting a focus window inside an image. The proxy provides the resource by connecting to the specified server and requesting the service on behalf of the client. A proxy may optionally alter the client's requests, retrieving a different subset of data from the server to that explicitly requested by the client. The proxy may also subset or reorder the server's response data in order to generate a response to its own client, possibly introducing data from its own cache. In some cases, the proxy may have sufficient cached data to serve a request without contacting the server. More generally, the proxy may start serving the client's request based its own cache contents, and later introduce additional content as it arrives from the server, so as to maximize the responsiveness of the network for interactive users. A JPIP proxy should typically realize the following services:

- understand the JPIP syntax, in order to check if the requested data are already stored in the cache or to cache JPEG2000 compressed data received from the server;
- edit, reformulate or resequence JPIP requests from its clients, in order to most efficiently utilize the available communication bandwidth with the server; and
- implement algorithms to efficiently serve its clients using cached data.

In Sections IV-A and IV-B we propose algorithms to enable these capabilities, while in Section V we show the gain obtained in remote browsing applications with a proxy that implements these capabilities.

A. Serving problem

The proxy's serving problem concerns the best order in which to deliver content in its cache to the clients. The solution adopted to solve this problem is similar to the approach used in [4].

Recalling Section III, we use the "EXTENDED HEADER" and "ALIGN" JPIP capabilities in the proxy's own requests to the server, so that the server's replies are aligned on layer boundaries and the number of layers in each message can be known by the client, as shown in Cases 2 and 3 of Figure 5.

Suppose that L_j^k, $k = 1, ..., K$ are the exact lengths of the quality layers for a precinct p_j, and let $\bar{L}_j^k$ denote the proxy's estimates of these lengths. In the event that the server sends exactly one layer of the precinct data-bin (i.e., one packet) in each message, the proxy can recover the original lengths, i.e., $\bar{L}_j^k = L_j^k$ for each layer k that the proxy has received and cached. More generally, the i^{th} message for precinct p_j contains layers s_j^i through to f_j^i, so that the proxy can only know the cumulative length $l_j^i = \sum_{k=s_j^i}^{f_j^i} L_j^k$. In this case, two possible approaches for estimating the individual lengths $\bar{L}_j^k$ are:

1) Set $\bar{L}_j^{s_j^i}$ through $\bar{L}_j^{f_j^i-1}$ equal to 0 and $\bar{L}_j^{f_j^i} = l_j^i$. In this case, the proxy will never send messages to its clients containing any of layers s_j^i through $f_j^i - 1$, without also including layer f_j^i, so the proxy's messages will aggregate quality layers to at least the same extent as the server's messages.

[1]The Origin Server interacts directly with the complete original image, stored on an appropriate file system.

2) Distribute the received bytes between the missed layers. For example, a uniform distribution policy would set $\bar{L}_j^k = l_j^i \big/ \left(f_j^i + 1 - s_j^i\right)$ for each $k \in \left[s_j^i, f_j^i\right]$. This potentially allows the proxy to send more uniform quality increments to its clients, which could result in lower distortion, but at the risk that layers are delivered only partially, so that clients cannot immediately utilize the transmitted data.

Independently from the approach adopted to form the length estimates $\bar{L}_j^k$, the following algorithm is used to sequence the cached precinct data. For each precinct p_j in the cache, let K_j be the value of the highest available quality layer with non-zero length. For each layer $k,\ k = 1, ..., K_j$, let D_j^k be the distortion and S_j^k the distortion-length slope for the layer, where S_j^k is defined as

$$S_j^k = \begin{cases} \frac{D_j^{k-1} - D_j^k}{L_j^k - L_j^{k-1}} & k = 1, ..., K_j \\ \infty & k = 0 \end{cases}$$

Recalling [4], the R-D optimal service policy is to serve the quality layers in decreasing order of the distortion-length slope S_j^k. Unfortunately, only during the encoding process is it possible to know exactly the values S_j^k, so the solution is to approximate these values with the distortion-length slope thresholds T_k, which were used during the encoding process to generate the quality layers. There is only one such threshold per quality layer, for the entire JPEG2000 code-stream. Moreover, these thresholds can be included in the header of the code-stream or the file which embeds it. We suppose that these values can be known at the proxy, or (in the worst case) guessed[2].

Since we are interested in transmitting only the precincts that affect the window required by a client, let us define the relevance function $r(p_j, W_c) = r_{j,c} \in (0, 1]$ for the precinct p_j with respect to window W_c. In particular the value of function $r(p, W)$ may be interpreted as the fraction of precinct p's subband samples which are involved in the reconstruction of window W. So, considering the precinct's relevance, the values that affect the distribution policy are the *weighted* rate-distortion slopes for non zero quality layers

$$\Gamma_j^k(W) = \begin{cases} r(p_j, W) T_k & \bar{L}_j^k > 0 \\ 0 & \bar{L}_j^k = 0 \end{cases}$$

With perfect length estimates, and a finely spaced collection of slope thresholds T_k, the optimal proxy serving policy is to deliver the cached contents of each precinct data-bin to the client in decreasing order of Γ_j^k. Figure 7 shows a comparison between the R-D performance obtained using the optimal policy described in [4], with clients connected directly to the server, and the proxy serving policy described here. The Figure shows results for both of the strategies described above for estimating cached layer lengths $\bar{L}_j^k$, from the messages which the proxy receives from its own server.

[2]The present algorithm relies only on knowledge of the ratios T_k/T_{k-1} between successive slope thresholds, which are often selected in predictable ways by compressors which generate reasonably disbursed quality layers.

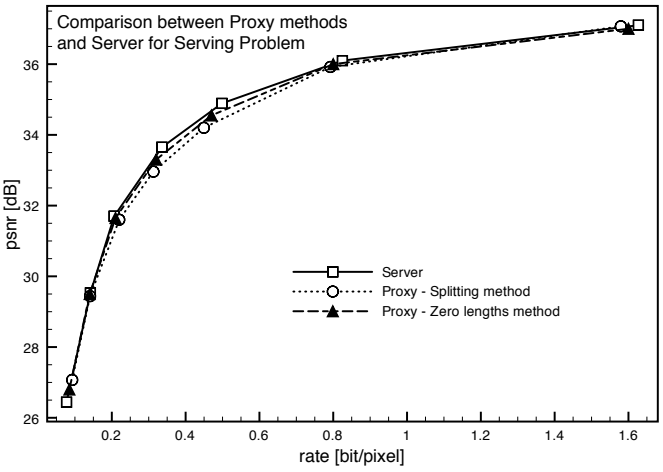

Fig. 7. Comparison between Server and Proxy for Serving Problem

B. Aggregation problem

In general, the proxy's aggregation problem concerns the way to aggregate individual client requests into requests which it forwards to the server. In this work we assume that the requests sent by the proxy to its server are drawn from the collection of original requests sent to the proxy by its clients. We write $W_c,\ c = 1, \ldots, C$ for the currently active client request windows, where C is the number of active clients. We also write W_R for the window request that the proxy chooses to send to its server at any given instant, where $W_R = W_c$ for some client index c. In our current implementation, the choice of request window W_R is re-evaluated only at those instants when a new reqest arrives from one of the clients, c. In this case, the proxy has only to decide whether W_R should be changed from its current value to W_c. This solution works well, so long as all clients issue requests at regular intervals, so that the proxy is able to regularly re-evaluate its choice of request window. Fortunately, this is exactly what happens in a typical HTTP-based JPIP communication session, since clients must issue requests with constraints on the number of bytes that should be returned, so as to ensure that interactive responsiveness is not damaged by a server/proxy flooding the underlying TCP channel.

The proxy's goal is to choose W_R in such a way as to maximize the rate at which the expected overall distortion experienced by its C clients can be reduced. This reduction in distortion occurs as the server's response data augments the proxy's cache representation, which may benefit more than one client in general.

It should be noted that the data received from the server may not match the proxy's expectations, since the server is generally free to send data in any order to the proxy. Moreover the proxy does not generally have sufficient information concerning the distortion associated with its current cache contents, or even the exact lengths of each precinct's quality layers, as noted earlier. Despite these shortcomings, the practical approach proposed here can be justified under

some reasonable assumptions, which we now expound.

The following distortion-rate model is commonly encountered in the literature and can be justified at least at high bit-rates:

$$D^* = Ue^{-gR}$$

Here, D^* measures Mean Squared Error (MSE) per sample and R is the encoded data rate, measured in bytes/sample. U and g are constants which generally depend upon the statistical properties of the source data, although we shall assume that the rate exponent g is a global constant. In the absence of more precise information, we apply this model directly to each precinct p_j, so that the total distortion D_j and total available code bytes

$$\mathcal{L}_j = \sum_{k=1}^{K_j} L_j^k$$

for that precinct may be related according to

$$D_j = A_j \cdot U_j \cdot e^{-g\mathcal{L}_j/A_j}$$

Here, A_j is the number of samples (area) associated with precinct p_j, allowing MSE D^* to be converted to total squared error D_j and data rate R to be converted to coded length $\mathcal{L}_j$.

Now suppose the proxy's current cache contents are described by the quantities K_j and $\mathcal{L}_j$, identifying the number of quality layers and the number of available data bytes for each precinct p_j. Also, suppose for the moment that the proxy also has access to the distortion-length slope $S_j^{K_j}$ associated with the K_j^{th} layer of precinct p_j. That is,

$$S_j^{K_j} = -\left.\frac{\partial D_j}{\partial \mathcal{L}}\right|_{\mathcal{L}=\mathcal{L}_j} = gU_j e^{-g\mathcal{L}_j/A_j}$$

Finally, suppose that the server uses the correct relevance factors $r_{j,R}$ to send additional data for each precinct $p_j \in W_R$, until the relevance weighted distortion-length slopes of all such precincts are reduced to a common value S_0. That is, the server sends an additional $\Delta\mathcal{L}_j$ bytes for each precinct $p_j \in W_R$ such that

$$gU_j e^{-g(\mathcal{L}_j+\Delta\mathcal{L}_j)/A_j} = \min\left\{S_j^{K_j}, \frac{S_0}{r_{j,R}}\right\}$$

This policy essentially describes the behaviour expected of the R-D optimal service policy described in [4], as well as the proxy's own serving algorithm, described in Section IV-A.

The actual value of S_0 depends upon the total amount of data the server will send before the request is changed again by the proxy. In the absence of any specific information, we shall assume that $\frac{S_0}{r_{j,R}}$ is much smaller than $S_j^{K_j}$ so that the $\min\{\}$ operator is not required to keep $\Delta\mathcal{L}_j$ positive. This allows us to deduce that

$$\Delta\mathcal{L}_j = \frac{A_j}{g}\ln\left(S_j^{K_j}\frac{r_{j,R}}{S_0}\right)$$

so the total number of bytes delivered by the server to reach slope target S_0 is

$$\Delta\mathcal{L} = \frac{1}{g}\sum_{p_j\in W_R} A_j \cdot \left[\ln\left(r_{j,R}S_j^{K_j}\right) - \ln(S_0)\right]$$

The increment of $\Delta\mathcal{L}_j$ bytes for precinct p_j reduces the distortion associated with that precinct by the amount

$$\begin{aligned}\Delta D_j &= A_j U_j e^{-g\mathcal{L}_j/A_j} \cdot \left[1 - e^{-g\Delta\mathcal{L}_j/A_j}\right] \\ &= \frac{A_j}{g}\left(S_j^{K_j} - \frac{S_0}{r_{j,R}}\right)\end{aligned}$$

It follows that the reduction in aggregate relevance-weighted distortion, taken over all client windows, is given by

$$\Delta D = \frac{1}{g}\sum_{c=1}^{C}\alpha_c \sum_{p_j\in W_c\cap W_R} r_{j,c}A_j\left(S_j^{K_j} - \frac{S_0}{r_{j,R}}\right)$$

Here, the factors α_c are client-specific weights, that can be used to account for clients with different levels of priority in the proxy's decision making process. For our experiments, all clients are considered equally important, with $\alpha_c = 1$ for all c.

Based on the above analysis, the proxy should set W_R to the request window which maximizes the ratio between ΔD and $\Delta\mathcal{L}$. Unfortunately, this ratio depends upon the unknown limit S_0. However, in the limit as the server's response to the chosen request becomes very long, S_0 becomes very small, so that

$$S_j^{K_j} - \frac{S_0}{r_{j,R}} \approx S_j^{K_j}$$

and

$$\ln\left(r_{j,R}S_j^{K_j}\right) - \ln(S_0) \approx \text{ constant}$$

This allows us to formulate the optimization objective as that of maximizing

$$J_R = \frac{\Delta D}{\Delta\mathcal{L}} = \frac{\sum_{c=1}^{C}\alpha_c\sum_{p_j\in W_c\cap W_R} r_{j,c}A_jS_j^{K_j}}{\sum_{p_j\in W_R}A_j}$$

So when a new request arrive from client c, the proxy has to compare J_R related to the window currently active with J_C related to the new candidate window W_C and set W_R to W_C if $J_C > J_R$.

V. Experimental Results

In order to evaluate the benefits given by a JPIP Proxy and to validate the policy proposed for the Aggregation Problem, two architectures has been compared: a Server-multi client and a Server-Proxy-multi client. Preliminary results have been obtain simulating remote browsing sessions with three clients. The optimal policy proposed in section IV-B has been compared to a FIFO algorithm in which each time that a new request of a window arrive at the proxy, W_R is automatically set to W_c. In order to show the difference between these two Aggregation policies the remote browsing sessions of Client 1 and 2 require windows spatially close, while Client 3 generates requests of windows in a different spatial area.

In the considered architectures the performance are highly affected by the rate of each communication link. Has been chosen to compare the Server-Clients architecture with all the links at the same rate equal to $r_s = R$ with a Server-Proxy-clients architecture where all the communication links (both

time	Client 1	Client 2	Client 3
0-5	1:(0,0)(640,480)	-	-
5-10	1:(0,0)(640,480)	1:(0,0)(640,480)	-
10-15	0:(100,100)(300,300)	1:(0,0)(640,480)	1:(0,0)(640,480)
15-20	0:(100,100)(300,300)	0:(100,250)(100,200)	1:(0,0)(640,480)
20-25	0:(200,200)(300,300)	0:(100,250)(100,200)	0:(600,300)(300,300)
25-30	0:(200,200)(300,300)	0:(250,250)(100,200)	0:(600,300)(300,300)
30-35	0:(0,0)(1280,720)	0:(250,250)(100,200)	0:(600,300)(300,300)
35-40	0:(0,0)(1280,720)	0:(0,0)(1280,720)	0:(600,300)(300,300)
40-45	0:(0,0)(1280,720)	0:(0,0)(1280,720)	0:(0,0)(1280,720)
45-50	0:(0,0)(1280,720)	0:(0,0)(1280,720)	0:(0,0)(1280,720)
50-55	-	0:(0,0)(1280,720)	0:(0,0)(1280,720)
55-60	-	-	0:(0,0)(1280,720)

TABLE I
DESCRIPTION OF REMOTE BROWSING SESSIONS

Server-Proxy link and Proxy-Clients links) have a rate equal to $r_p = NR$ where N is the number of clients. Although this comparison seems unfair, it gives the possibility to show the increase in efficiency given by the use of a Proxy Server.

Table I shows the description of the remote browsing sessions simulated for each client, where the notation $R : (r_x, r_y)(s_x, s_y)$ means that a client request the resolution R, where in the tests performed only the full resolution (R=0) equal to 1280x720 pixels and the half resolution (R=1) has been considered, with a Focus Window starting at position (r_x, r_y) and with size (s_x, s_y).

Figure 8 shows the R-D comparison between the Server-Clients architecture (Server), Server-Proxy-Clients using a FIFO algorithm for the Serving problem (Proxy FIFO) and the optimal algorithm presented in section IV-B (Proxy RD). As shown in Figure 8 the Architecture with a Proxy Server can greatly improve the performance compared with a simple Server-Clients model. The gain depends on the quantity of data stored in the cache. So, for example in the beginning of remote browsing session of Client 1 the Proxy has no data in the cache and the performance are comparable with the Server-Clients model. As soon as the Proxy receive data from the Server, it can reuse this data to serve other Clients without contact the Server. This is the reason because the gain obtained with Client 2 is higher than the gain for Client 1, since when the Client 2 starts the session the Proxy has already stored a good amount of data in the cache. From the comparison between the optimal algorithm proposed in section IV-B and the FIFO algorithm can be notice as the optimal algorithm increase the overall performance. This is obtained giving "high priority" to Client 1 and 2 because they request spatially adjacent windows. The result is that Client 1 and 2 increase the performance compared to FIFO algorithm, but decrease the performance of Client 3. In some sense the FIFO algorithm seems more fair, but it shows lower performance compared to optimal algorithm.

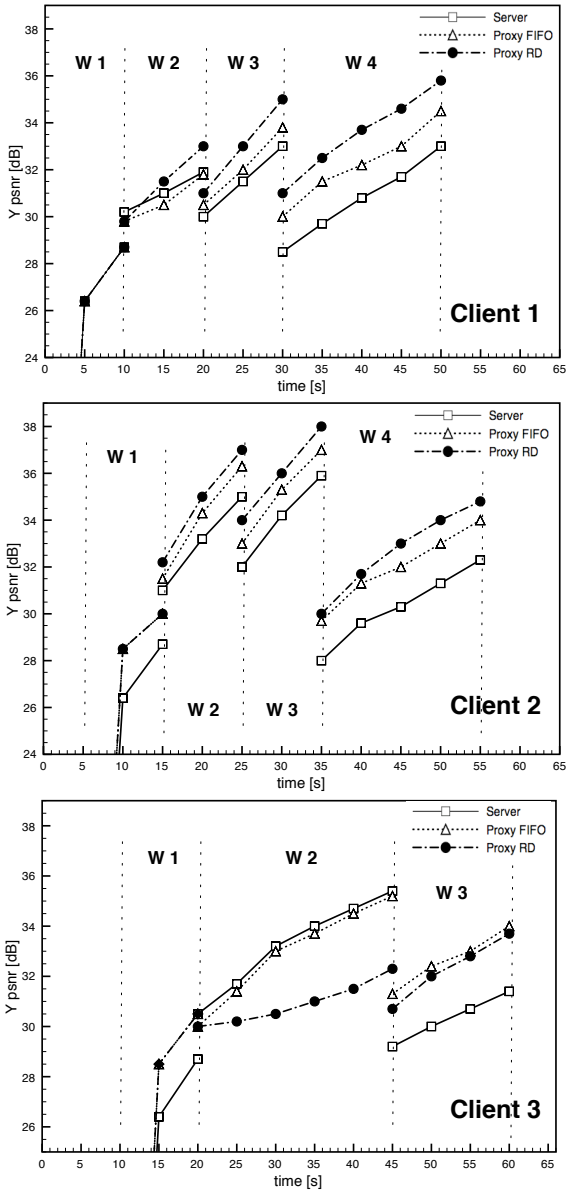

Fig. 8. Remote browsing simulation with 3 clients

VI. CONCLUSIONS

This work proposed a JPIP Proxy Server for remote browsing applications. In particular has been shown as the architecture using the Proxy can improve the performance of the whole system, through an efficient use of the communication link with the server. Moreover the R-D optimal approach for Serving Problem shown better performance compared to other algorithms like FIFO, especially when multiple clients require Focus Window spatially adjacent. Another important consideration is that the performance could be further increase if the Proxy Server starts with a non-empty cache, for example retrieving data from previous sessions.

Possible extension of the proposed work could include architectures with multiple Proxies and Servers connected each other, connection between Proxy and Clients using a shared channel instead of dedicated channels.

REFERENCES

[1] S. Deshpande and W. Zeng, "Scalable Streaming of JPEG2000 Images using Hypertext Transfer Protocol," in *Proc. ACM Multimedia*, 2001.

[2] "ITU-T Recommendation T.808 — ISO/IEC 15444-9:2004, Information Technology - JPEG 2000 image coding system - Part 9: Interactivity tools, APIs and protocols," Tech. Rep.

[3] D. Taubman and R. Prandolini, "Architecture, Philosophy and Performance of JPIP: Internet Protocol Standard for JPEG2000," in *Proc. VCIP 2003*.

[4] D. Taubman and R. Rosenbaum, "Rate-Distortion Optimized Interactive Browsing of JPEG2000 images," in *Proc. ICIP 2003*.