Multi-Source Video Streaming Suite

Pablo Rodríguez-Bocca[1,2], Gerardo Rubino[2], and Luis Stábile[2]

[1] Instituto de Computación - Facultad de Ingeniería
Universidad de la República,
Julio Herrera y Reissig 565, 11300,
Montevideo, Uruguay
prbocca@fing.edu.uy
[2] Inria/Irisa
Campus universitaire de Beaulieu
35042 Rennes, France

Abstract. This paper presents a method for the distribution of video flows through several paths of an IP network. We call the approach *multi-source* focusing on the fact that, from the receiver's point of view, the stream is obtained from several different sources. A typical use of our procedure is in the design of a P2P network for streaming applications. We implemented a tool in VLC which can send video streaming from multiple sources. In this paper, we describe the method together with the design decisions and the operation characteristics of the tool. Finally we consider, at the receiver side, the perceived quality, automatically measured using the PSQA methodology. We illustrate the work of the tool by means of some tests.
Keywords: multi-source, Internet Protocol, VLC, PSQA

1 Introduction

The main objective of our tool is the distribution of video flows through several paths of an IP network. We call the approach *multi-source* focusing on the fact that, from the receiver's point of view, the stream is obtained from several different sources. The idea is to resist to the possible failures of the servers distributing somehow the associated risks. Therefore, multi-source streaming techniques are used typically in P2P networks, where the peers become often disconnected.

Previous studies (see [1]) show that in 30-80% of the cases there is at least a better end-to-end path, from the quality perspective, for a transmission. In the particular case of video flows, it is then easier to recover from isolated losses than from consecutive ones [2]. Another advantage of this approach is that it helps to balance the load on the networks (we are not going to address this issue in this paper). The basic difference between the multi-path and multi-source concepts is that in the former case we consider a single source sending data to the destination following different paths. The main difficulty here is the routing aspects: we must specify and control which path must be followed by which packet in the flow. The multi-source approach implies that there are multiple and independent sources for the signal, and that some general scheme allows

the receiver to get the stream from a set of servers. Since they are different, the path that the packets will follow would be a priori different, without any need of acting on the routing processes. Of course, in practice, the different paths could share some nodes, which are good candidates to become bottlenecks of the communication system. Detecting such cases is an active research area [3].

A few studies consider the implementation needed to split and to merge stream content, especially when it is live video streaming. For instance, Nguyen and Zakhor [4] propose to stream video from multiple sources concurrently, thereby exploiting path diversity and increasing tolerance to packet loss. Rodrigues and Biersack [5] show that parallel download of a large file from multiple replicated servers achieves significantly shorter download times. Apostolopoulos [1,6] originally proposed using striped video and Multiple Description Coding (MDC) to exploit path diversity for increased robustness to packet loss. They propose building an overlay composed of relays, and having each stripe delivered to the client using a different source. Besides the academic studies, some commercial networks for video distribution are available. Focusing in the P2P world (our main application here), the most successful ones are PPlive [7], SopCast [8], and TVUnetwork [9].

The paper is organized as follows. Section 2 introduces all relevant concepts for our project. In Section 3, a conceptual architecture of the components and the complete solution is presented. It consists of modules added to the open code VLC project of VideoLAN [10]. In Section 4 some first experimental results are introduced. The main contributions of this work and conclusions are in Section 5.

2 Multi-Source Streaming

We will focus on the case of a P2P architecture where the video flow is decomposed into pieces and sent through the network. Each node receives the flow from different sources and builds the original stream before it is played. At the same time, it will, in general, send each of the substreams to a different client, acting then as a server. At the beginning, the (single) initial stream is split into several substreams, according to some specific scheme.

With these goals, we need the two basic following services: (i) a flexible "splitter" where the original stream is decomposed into different substreams, and (ii) a module capable of reconstructing the original stream from the set of substreams, or a degraded version of it if some of the substreams had losses, or is simply missing. Associated with these services we want to design a very flexible transport scheme, allowing to transport the original information plus some redundancy, with high control on which server sends which part of the flow and of the redundant information (see below). We also want to be able to measure, at the receiver side, the perceived quality, using the PSQA methodology.

Let us now briefly describe these different components of the project. The splitter must be able to obey a general scheme where we decide how many substreams are to be used, and which of the frames are to be sent through which of the substreams. This generality allows us to send, for instance, most of

the frames through the highest performing nodes in the network, or to balance the load among the different components, or to adopt a scheme where the type of frame is used to take the routing decision. It must also be possible to send an extra fraction r of the original stream, or of a specific population of the frames (for instance, the I frames) again according to any pre-specified scheme. If $r = 0$ there is no redundancy at all. If $r = 0.2$ we send 20% of supplementary redundant data, and $r = 1$ means that the original stream is actually sent twice to the receiver(s) We not consider $r > 1$ because, in a real system, this would mean too much bandwidth consumption. At the client side, we must be able to reconstruct the original stream if we receive all the frames, but also if only a part of the stream arrives; this typically happens when some of the servers fails (that is, it disconnects from the P2P network). Last, the client must be able to identify the lost frames and to compute statistics from them. We want also that the client computes the value of the quality of the received stream, as evaluated by the PSQA tool, for instance for auditing the quality of the transmission system, or for controlling purposes.

2.1 The PSQA Technology [11]

PSQA stands for Pseudo-Subjective Quality Assessment. It is a technique that allow the building of a function mapping some *measurable* parameters chosen by the users into a numerical assessment of the perceived quality. The measures are taken on the client side, and a good example is the end-to-end loss rate of frames. The mapping is done by showing many sequences to real human users, where these parameters took many different values, and performing subjective tests in order to evaluate their quality values. Then, after statistically filtering the results provided by these tests, a statistical learning tool is used that allows to build the PSQA function. To use PSQA on the client side, we must be able to measure the parameters chosen at the beginning of the process, and then to call the function. The latter is always a rational function of the parameters, whose value can be obtained very quickly, allowing PSQA to operate, if necessary, in real time. The parameters are chosen at the beginning of the process. They can be qualitative or quantitative. Their choice depend on the network and/or the application at hand, and of course the resulting PSQA procedure will be better if the a priori choice is as relevant as possible.

We know from previous work on PSQA that the loss process is the most important global factor for quality. In this paper, we consider the loss rates of video frames, denoted by LR, and the mean size of loss bursts, $MLBS$, that is, the average length of a sequence of consecutive lost frames not contained in a longer such sequence. See [11] and the references therein for more details about PSQAs.

3 Design

This section will describe the main design decisions. There are K servers and each server is identified by an index between 1 and K. See Figure 1 for an illustration.

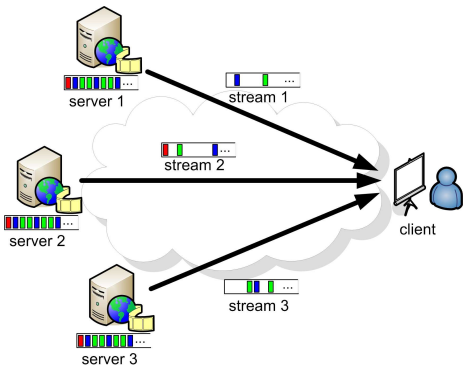

Fig. 1. The multi-source scheme

3.1 Global Architecture

The implementation consists of three VLC modules, one at the server side and two at the client side. They are called `msource-server`, `msource-bridge` and `msource-client` (see Figure 2).

The module `msource-server`, located at the server, decides which frames are to be sent to the client. It builds a substream for audio and another one for video. The basic constraint to satisfy is, of course, that each frame in the original stream must be sent at least once by one of the servers. Once the frames selected by `msource-server`, the module `standard` of VLC sends them to the client.

At the client's side, there are K modules `msource-bridge`, one per server and each with its own buffer. The kth one receives the frames sent by server k. The last module, `msource-client`, receives the flows sent by the K modules `msource-bridge` and reconstructs the stream. This task consists of ordering the frames according to their decoding time (called DTS). The output of a module `msource-client` can be stored on the local disk, sent somewhere through the network (both tasks are performed by the module `standard`, which is a part of the VLC package) or played by the client.

The ideal case is when all the servers start their transmissions simultaneously and keep synchronized, but this never happens in practice. It means that the system must handle the possible shift between the servers. Moreover, since we

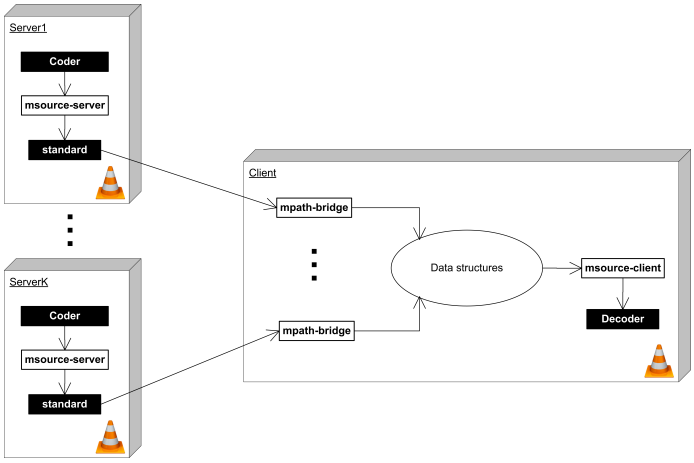

Fig. 2. Architecture multiple-source in VLC

don't assume that the servers are identical, we implement a strategy allowing to control the bandwidth used by each of them.

3.2 Basic Server Algorithm

The strategy used by the servers allows to control the bandwidth used by each of them, for instance, to be able to send more information through the most reliable one, or through the one having the best performance. This control is implemented by acting on the percentage of frames of each type that the server must send. We must decide which fraction $p_k^{(c)}$ of the class c frames, $c \in \{\text{I, P, B, A}\}$ [3] is going to be send through server k. We must then have

$$\sum_{k=1}^{K} p_k^{(c)} = 1, \quad c \in \{\text{I, P, B, A}\}. \tag{1}$$

The sending algorithm works as follows. All the servers run the same pseudo-random number generator and build a sequence of pseudo-random reals uniformly distributed on $[0, 1]$. Not only they run the same generator but they use the same seed. In this way, they obtain the same sequence of real numbers $(u_1, u_2, \cdots)$ behaving as a realization of the corresponding sequence of i.i.d. random variables uniformly distributed on $[0, 1]$.

Now, look at $(p_k^{(c)})_{k=1,\cdots,K}$, for fixed c, as a probability distribution, and call $X^{(c)}$ the corresponding random variable. It can be sampled from an uniform

[3] In MPEG, basically there are three classes of video frames (I, P and B), and one class of audio frames (A).

random variable U using the classical algorithm given by

$$X^{(c)} = k \Longleftrightarrow P_{k-1}^{(c)} \leq U < P_k^{(c)}, \tag{2}$$

where $P_j^{(c)} = p_1^{(c)} + \cdots + p_j^{(c)}$, $j = 1, \cdots, K$, and $P_0^{(c)} = 0$.

Now, let f_n be the nth frame of the stream, $n \geq 1$, and let $c_n \in \{\text{I, P, B, A}\}$ be the class of f_n. Any of the K servers will then run the same algorithm. They all sample the random variable $X^{(c)}$, and obtain the same value s_n, with $1 \leq s_n \leq K$. That is, they all look at the type of f_n, and use (2). Server s_n sends f_n and the remaining servers don't. This construction guarantees that one and only one server sends each frame, and since s_n behaves as a realization of random variable $X^{(c)}$, after sending many frames the fraction of type c frames sent by server k will be close to $p_k^{(c)}$. Observe that this method allows to control the bandwidth of each server in a scalable way (there is no limit on the number of servers nor on the distribution of the load among them).

3.3 Controlling the Redundancy Level

With our method we can send some redundancy to the client. This, together with our splitting procedure, adds robustness to the system, in order to face the problem of possible server failures (recall that we call failure any event causing the server to stop sending, for instance, because it simply left the network). As we will see in next subsection, redundancy also allows us to design a simple solution to the problem of synchronizing. We describe here the way we control the redundancy level in the system.

We allow the system to send redundant frames up to sending again the whole stream, and we provide a precise mechanism to control with high precision the redundancy level and the distribution of the supplementary load among the K servers. For this purpose, we implement a system where a given frame is sent either once or twice, and in the latter case, by two different servers. Our method allows to control the redundancy level per class, for instance, for each class of frames (I, P, B). We denote by $r^{(c)}$ the fraction of supplementary class-c frames that will be sent to the client (by the set of servers). So, each server k must decide if it sends frame f_n as the "original" frame, as a copy for redundancy, or not at all. The procedure described before allows to choose the server that sends the frame as the original one. For the redundacy, the implementation is also probabilistic. We proceed as follows. Together with the stream of pseudo-random numbers used to make the first assignment we described before, the set of servers build a second sequence $(v_1, v_2, \cdots)$ with the same characteristics (the same for all servers, the same seed). Suppose that frame f_n is of class c and that it is assigned to some other server j, $j \neq k$. Then, using the second sequence $(v_1, v_2, \cdots)$, server k samples a second random variable with values in the set $\{1, 2, \cdots, K\}$ to decide at the same time if a redundant copy of the frame is to be send and if it must send it. Let us call $Y_j^{(c)}$ this random variable. Its distribution

is the following: $\Pr(Y_j^{(c)} = j) = 1 - r^{(c)}$ and if $m \neq j$,

$$\Pr(Y_j^{(c)} = m) = \frac{p_m^{(c)}}{\sum_{h:\, h \neq m} p_h^{(c)}} r^{(c)} = \frac{p_m^{(c)}}{1 - p_j^{(c)}} r^{(c)}.$$

If $Y_j^{(c)} = j$, no redundancy is sent. If $Y_j^{(c)} = m$, $m \neq j$, server m is the only one to send the frame as a redundant one.

3.4 Client Behavior

The client must reconstruct the stream using the flows received from the different servers. It will work even if some of the servers are missing (failed). Each server sends its streams marked with a time stamp indicating its playtime with respect to a specific reference (in VLC, 1/1/1970). Assuming all the servers synchronized, the algorithm is simple: it consists of selecting as the next frame the one having the smallest value of playtime. The problem is the possible shift between the servers (whose locations are different in the network). This can be due to the conditions encountered by the packets in their travelling through the network, or by the processing of the frames at the servers themselves. The key idea for knowing this shift in the transmission time of the servers consists of using the redundant frames.

First, let us look at the identification of the redundant frames. Each frame is sent with a time stamp corresponding to its playtime at the client side (computed by VLC). When receiving it, a Message-Digest algorithm 5 (MD5) [12] is computed and a dictionary is maintained to see if an arriving frame has already been received (necessarily by a different server). If the frame arrives for the first time, we store its MD5 together with its time stamp. Assume now that the same frame arrived from two different servers i and j, and call τ_i and τ_j the corresponding time stamps. The difference between these values is the (current) shift between the two servers. We denote it by Δ_{ij}, that is, $\Delta_{ij} = \tau_i - \tau_j$. Let us denote by Δ the $K \times K$ matrix (Δ_{ij}) where we define $\Delta_{ii} = 0$ for all server i. Observe that $\Delta_{ji} = -\Delta_{ij}$. Following these observations, we maintain such a matrix, initialized to 0, and we modify it each time a new redundant frame is detected (actually we do it less often, but this is a detail here). Observe that rows i and j in the matrix, $i \neq j$, are obtained by adding a constant element per element (or, equivalently, see that $\Delta_{ij} = \Delta_{ik} + \Delta_{kj}$). This constant is precisely the delay between the corresponding servers. The same happens with the columns. All this in other words: if we receive $K - 1$ redundant pairs corresponding to $K - 1$ different pairs of servers, we can build the whole matrix.

Each time we update matrix Δ, we can compute the index d of the most delayed server (if any), by choosing any row[4] in Δ and by computing its smallest element. Then, using for instance row 1, $d = \text{argmin}\{j : \Delta_{1j}\}$. When the client now looks for the next frame to choose, it scans the K buffers corresponding

[4] Actually, we can choose any row if all the entries of Δ have been computed; otherwise, we choose the row having the largest number of computed entries.

to the K servers. Let us denote by τ_k the time stamp of the first frame in the kth buffer (assume for the moment that no buffer is empty). Then, we first make the correcting operation $\tau'_k = \tau_k + \Delta_{dk}$, that is, we synchronize with the time of the most delayed server, and then we look for the server m where $m = \text{argmin}\{k : \tau'_k\}$. Next frame to be played will be the one in head of buffer m. This works as long as there are frames in buffer d. If it is empty after a play, we must wait for an arrival there, because we have no information about which server will be sending next frame to play. Of course, we must wait until some time out because if server d for some reason stopped sending, we block the system if we remain waiting for its frames. After some amount of time, we move to the frame having the smallest time to play using the previous procedure.

Observe that, at the price of an extra computation at the client's side, we are able to synchronize efficiently the substreams without any signalling traffic, and using the same data that protects the system against failures.

4 Evaluation and first results

For testing the correctness of our prototype, we implemented a program that collects complete traces of the sequences of frames sent by the servers and received by the client. These traces allows us to determine, for each frame, which server sent it and if it was played of not by the client. The program also collects some other data as the amount of frames sent, the used bandwidth, etc.

4.1 Testing the bandwidth used

The goal of the tests we will describe here is to measure the bandwidth used by the servers. This can be compared with the values provided by a theoretical analysis of the system, to check the consistency of the implementation. We will use two different models for distributing the load among the servers. The first one consists of sharing it uniformly among the K servers, that is, all of them send the same fraction $1/K$ (quantitatively speaking) of the global stream. In the second model, we use a geometric distribution: server i sends $1/2^i$th of the stream, for $i = 1, 2, \cdots, K-1$ and server K sends the fraction $1/2^{K-1}$.

Consider the uniform case, in which we send the stream with a redundancy level of r. If $BW_{K,i}^{unif}$ denotes the bandwidth used by server i, then clearly $BW_{K,i}^{unif} = (1+r)/K$. The case of our geometric load is a little bit more involved. If $BW_{K,i}^{geo}$ is the bandwidth used by server i in this case, we have

$$BW_{K,i}^{geo} = \begin{cases} \dfrac{1}{2^i} + \dfrac{r}{2^i}\left(1 - \dfrac{1}{2^i}\right) & \text{if } i \neq K \\ \dfrac{1}{2^{K-1}} + \dfrac{r}{2^{K-1}}\left(1 - \dfrac{1}{2^{K-1}}\right) & \text{if } i = K \end{cases}, \quad K \geq i \geq 1. \tag{3}$$

In our tests, we used the value $r = 0.5$. In Table 1 we show the Mean Squared Error between the bandwidth measured during our experiments and the theoretical values in the two distribution models considered, uniform and geometric.

We evaluated the used bandwidth by extracting information automatically computed by the standard VLC modules. This consists of multiplying the number of frames sent in each class (I, P, ...) by the average size of these frames, and then by dividing by the used time. We then get an approximation of the used bandwidth. This, plus the losses (see next subsection) explains the differences between expected and observed values. Observe that we sum over all servers, which means that the effects of the randomness used are not responsible of any part of the observed differences.

K	Uniform	Geometric
1	0.00000	0.00000
2	0.00005	0.00005
3	0.00013	0.00322
4	0.00006	0.00310
5	0.00008	0.00243
6	0.00005	0.00207
7	0.00006	0.00171
8	0.00004	0.00149
9	0.00009	0.00134
10	0.00011	0.00125

Table 1. Mean Squared Error between theoretical and observed values of the used bandwidth, as a function of the total number K of servers. The error is computed summing on i, the server index, from 1 to K.

4.2 Measuring losses

In our preliminary prototype, there are some frame losses at the beginning of the transmission, because we assume no specific effort is made to synchronize the K servers (this extreme situation is considered for testing purposes). There is, then, a transient phase during which the system will loose information until there have been enough redundancies allowing to synchronize the flows using the procedure described before. Then, during the transmission, in other stressing experimental situations, some of the servers may have very few frames to send, which can make the synchronization process again slower to converge, until redundant frames are finally sent.

We computed the loss rates by comparing the sent and received frame sequences. We arbitrarily eliminated the first parts of the flows until observing 50 consecutive frames correctly sent, because this implies in general that the K servers are synchronized. Again, the goal is to check that even using this rough testing procedure, the observed loss ratios are small enough. We used the same loads as in Subsection 4.1. In Table 2 we show the observed loss rates, during $2K$

experiments using k servers, $1 \leq k \leq K$, and both the uniform and geometric load distributions.

K	loss rate (uniform load)	loss rate (geometric load)
1	0.0000	0.0000
2	0.0049	0.0049
3	0.0070	0.0080
4	0.0083	0.0066
5	0.0080	0.0070
6	0.0080	0.0072
7	0.0083	0.0220
8	0.0093	0.0186
9	0.0090	0.0182
10	0.0129	0.0222

Table 2. Estimated loss rates after synchronization, as a function of the number K of servers used, for the two load models considered

Observe the fact that in the geometric model, we measure some "peak" values of the loss rates for high values of K. This is due to the fact that in this load model there are servers that send a small number of frames. Specifically, all the servers with index $i \geq 7$ send $(1/2^6)$th of the original video. These servers never succeed in synchronizing because of the lack of redundant frames sent. This implies that the decoding time of the frames sent by these servers will not be correctly computed because the shifts will not been known by the system. The final consequence of this is that those frames will be discarded.

4.3 Perceived quality

In this last subsection, we consider the perceived quality of the reconstructed stream. In the experiments mentioned here, the goal is to test the consistency of the system, and not its performances (this will be the object of further studies). However, we underline here the way the connection is made with the quality as perceived by the user.

The PSQA technology provides us with a function $Q = Q(LR, MLBS)$. In Figure 3 we plot the curve obtained following the PSQA procedure (see [11] for a general presentation of the PSQA approach and metrics such as LR and MLBS; see [13] for an example of application). First, observe that the quality is much more sensitive to losses than to the average size of the loss bursts. If we want to estimate the quality as perceived by the user, we must just measure the LR and $MLBS$ and call the Q function using the measured values as input data. For instance, consider the loss rates obtained using the uniform load model,

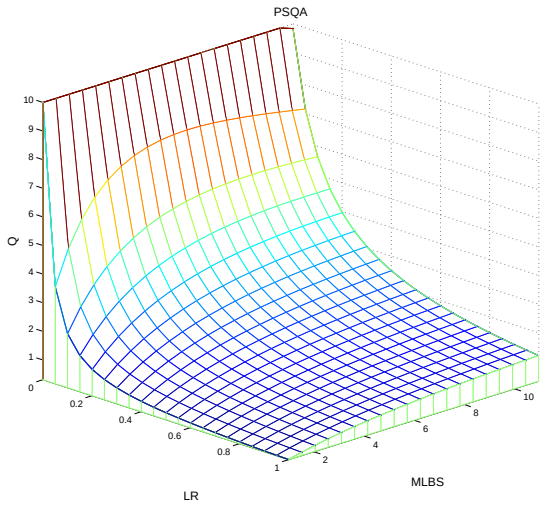

Fig. 3. The PSQA function used

as shown in Table 2. Figure 3 shows that the worst case is when MLBS = 1. Using this value and the loss rates given in the first column of Table 2, we obtain, as perceived quality estimates, values going from 10 (maximal quality) to approximately 9, which is almost maximal. For the geometric load (somehow an extreme situation), we observe loss ratios up to 0.0222, which in the worst case of MLBS = 1, translates into a quality level of about 5 (this corresponds to the characterization "fair" in the subjective testing area). The reason was already commented: the servers having little to send have no time to synchronize. This is telling us that the losses due to the synchronization problem we observed, even if the situation which generated them is a little bit too pessimistic, should be diminished. This is commented in the last concluding section.

5 Conclusions

This work shows that a multiple source method for distributing live video can be efficiently implemented with a fine control on its redundant capacities. We also show that we can connect such a transport system with a tool allowing to measure the perceived quality as seen by the end user (the client), which can be used for controlling purposes.

One of the results we obtained is that the redundancy of our system not only provides robustness against servers' failures (typically, due to nodes leaving the networks) but also allows to implement a method for synchronizing the flows without any other signalling process.

The first measures we did show also that the present prototype needs some improvements to reduce the losses due to the possible lack of synchronization (even if they are not very important in number). In order to diminish these losses, we are currently working on adding extra redundancy for a better synchronization (for instance, by sending at the beginning, all the frames by all the servers, during some pre-specified time or until a pre-chosen number of frames have been sent). Also, when a server has almost no frame to send, we also can force it to send redundant data, again for allowing the client to estimate the drifts with accuracy enough.

One of the tasks needed for further work is to explore the performances of our methods in different failure scenarios. The goal of this study will be to explore the appropriate values of the number K of servers to use, together with the best redundancy level r. After such a performance evaluation analysis, the search for optimal tuning parameters can be considered.

References

1. Apostolopoulos, J., Trott, M.: Path diversity for enhanced media streaming. IEEE Communications Magazine **42**(8) (2004) 80–87
2. Apostolopoulos, J.: Reliable video communication over lossy packet networks using multiple state encoding and path diversity. In: Proc. Visual Communication and Image Processing, VCIP '01. (2001) 392–409
3. Rubenstein, D., Kurose, J., Towsley, D.: Detecting shared congestion of flows via end-to-end measurement. IEEE Trans. on Networking **3** (2003)
4. Nguyen, T.P., Zakhor, A.: Distributed video streaming over Internet. In Kienzle, M.G., Shenoy, P.J., eds.: Proc. SPIE Vol. 4673, p. 186-195, Multimedia Computing and Networking 2002, Martin G. Kienzle; Prashant J. Shenoy; Eds. Volume 4673 of Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference. (2001) 186–195
5. Rodriguez, P., Biersack, E.W.: Dynamic parallel access to replicated content in the internet. IEEE/ACM Trans. Netw. **10**(4) (2002) 455–465
6. Apostolopoulos, J., Wee, S.: Unbalanced multiple description video communication using path diversity. In: In IEEE International Conference on Image Processing. (2001) 966–969
7. PPLive Home page. http://www.pplive.com (2007)
8. SopCast - Free P2P internet TV. http://www.sopcast.org (2007)
9. TVUnetworks home page. http://tvunetworks.com/ (2007)
10. VideoLan home page. http://www.videolan.org (2007)
11. Rubino, G.: Quantifying the Quality of Audio and Video Transmissions over the Internet: the PSQA Approach. In: Design and Operations of Communication Networks: A Review of Wired and Wireless Modelling and Management Challenges. Edited by J. Barria. Imperial College Press (2005)
12. IETF Network Working Group: The MD5 Message-Digest Algorithm (RFC 1321) (1992)
13. Bonnin, J.M., Rubino, G., Varela, M.: Controlling Multimedia QoS in the Future Home Network Using the PSQA Metric. The Computer Journal **2**(49) (2006)