BitTorrent based Transmission of Real-Time Scalable Video over P2P Networks

Pedro L. Rodrigues
Departamento de Engª. Eletrotécnica
Instituto Superior de Engenharia, UAlg
Faro, Portugal
pedro.larguito.rodrigues@gmail.com

Jânio M. Monteiro
INESC Inovação, Lisbon, Portugal &
Instituto Superior de Engenharia, UAlg
Faro, Portugal

Abstract— The number of software applications available on the Internet for distributing video streams in real time over P2P networks has grown quickly in the last two years. Typical this kind of distribution is made by television channel broadcasters which try to make their content globally available, using viewer's resources to support a large scale distribution of video without incurring in incremental costs. However, the lack of adaptation in video quality , combined with the lack of a standard protocol for this kind of multimedia distribution has driven content providers to basically ignore it as a solution for video delivery over the Internet. While the scalable extension of the H.264 encoding (H.264/SVC) can be used to support terminal and network heterogeneity, it is not clear how it can be integrated in a P2P overlay to form a large scale and real time distribution. In this paper, we start by defining a solution that combines the most popular P2P file-sharing protocol, the BitTorrent, with the H.264/SVC encoding for a real-time video content delivery. Using this solution we then evaluate the effect of several parameters in the quality received by peers.

Keywords- Peer-to-Peer (P2P), Video, Scalable Video Coding (SVC), Real-Time, BitTorrent

I. INTRODUCTION

The massive utilization of video services over the Internet like YouTube, live Internet video, online video purchases and rentals, webcam viewing and web-based video monitoring has increased the global traffic of Internet video so significantly, that for the first time in ten years P2P file-sharing traffic is no longer the largest Internet traffic type in the Internet. Video took its place during 2010, with 40% of the overall traffic, with some forecasts predicting that it will reach 90% in 2015, as described in [1].

This growth partially results from the continuous increment of the utilization of P2PTV software applications like TVUPlayer or PPStream, that distribute video content in real time over P2P networks. Typical this kind of distribution is made by TV broadcast channels from all over the world trying to make their contents globally available. By using the peer's computational and network resources, they are able to distribute their video content to a much larger number of receivers, without incurring in additional infrastructure costs.

However, all these software applications suffer from very similar problems, including: lack of adaption to bandwidth fluctuations; lack of support for different type of terminals; low quality of the multimedia streams due to the lack of support of Quality of Service (QoS) in the Internet; and finally they are proprietary solutions without a defined standard.

In terms of video the Scalable Video Coding (SVC) extension [2] of the H.264/AVC standard enables the transmission and decoding of several layers with different temporal, spatial or signal-to-noise ratio resolutions. The scalability property of SVC supports the splitting of the main bit stream into several sub-streams, forming a base layer, fully compatible with H.264/AVC devices, and several enhancement layers. These properties support the adaptation of the video quality to different bit rates and distinct terminal capabilities. When combined with an adequate protection mechanism it also enables a graceful degradation of quality for the transmission over loss prone channels [3].

In terms of P2P, the BitTorrent protocol is the most popular P2P file-sharing protocol used on the Internet. In 2008 P2P file-sharing traffic was the largest type of traffic on Internet and 70% of this traffic was estimated to be made by BitTorrent clients [4]. However, although it is extensively used in the transmission of non-real time data, its adaptation for the transmission of real-time video is still an open issue.

Given these considerations, the aim of this paper is to evaluate how to combine the P2P file-sharing protocol BitTorrent with the scalable extension of the H.264 encoding (SVC) [2] for the support of real-time video delivery. The rest of this paper is organized as follows. Section II describes some of the main features of the BitTorrent protocol. Section III presents the changes required to enable the transmission of H.264/SVC over IP networks taking into account its scalability properties. Section IV describes the implemented simulator and the improvements made to the BitTorrent protocol to support a real-time transmission of SVC. Section V presents the obtained simulation results. Finally, section VI concludes the paper pointing out future developments that follow from this work.

II. BITTORRENT PROTOCOL

The BitTorrent protocol [5] was designed to distribute large files in smaller pieces using a mutual distribution method between a group of peers called a swarm. To download a file using BitTorrent protocol, besides requiring a BitTorrent client installed, a torrent file is needed. The torrent file contains very important information required to download the content files, including the fixed piece size in which the content files will be chopped and the URL of the tracker of the swarm. The tracker

is built from a web server and monitors all the peers in the swarm. Each peer contacts the tracker to get a list of peers in the swarm and, after that, starts communicating directly with the other peers, i.e. without intervention of the tracker. Active peers that have a complete copy of the content are called seeds and peers still downloading the content are called leechers [5].

Each peer requests pieces to download from its neighbor peers in a random order. Each time a peer has successfully downloaded a piece of the file, it announces it to the other connected peers within the swarm.

The purpose of each peer is to maximize its piece download rate in a reciprocal manner. Several policies are implemented in the protocol to achieve fairness, provide incentives for mutual exchange, avoid overloading and find better peers with which to exchange pieces. Each peer should avoid being overloaded by requests for pieces. For this reason and for good TCP performance, each peer limits the number of simultaneous active connections, typically to four different peers [6]. Then the choke/unchoke mechanism is put into place and the active connections are placed in the unchoked state, while the other neighbor connections are placed in the choked state. To avoid repetitive changes of choked state, called as fibrillation, this process is limited to occur in intervals of 10 seconds. Exchanges with neighbors should be fair, that is, each peer should reciprocate by supplying pieces to peers that also provide downloads to it. Finally, each peer should try to connect other peers periodically to see if their download rate is better than the current ones in its active group of peers.

To achieve the fairness and performance goals, each peer follows several strategies. First, it uses a rarest-first prioritization of pieces to download by selecting pieces that have been least downloaded, increasing the probability of other peers becoming interested in exchanging pieces with it. Second, it favors peers with higher capacity of transferring information, based on previous data transfer rates with each neighbor peer. Third, it tests other peers periodically to see if better transfer rates can be obtained. This optimistic unchoking mechanism selects an interested random peer every 30 seconds.

Additionally, the BitTorrent protocol uses a called tit-for-tat strategy in which the downloading peer chooses other peers based on the capacity of that peer to download the piece he is interested in. By sending pieces to a group of peers, each peer increases the chances of receiving pieces from them. A peer indicates it is either interested or uninterested in receiving exchanges from a neighbor peer in the swarm.

When data is being transferred, downloading peers queue several piece requests at once to get good transport performance, this technique is called pipelining.

III. Adaptation of H.264/SVC Streams for the Transmission over IP Networks

In SVC, each NAL unit has a direct association with three basic scalable dimensions: spatial (i.e., resolution), temporal (i.e., frame rate) and quality (i.e., SNR), which in turn can be identified using three identifiers: dependency_id, temporal_id, and quality_id. These identifiers can also be referred as a (D,T,Q) tuple, as described in [4].

In the first part of this study, transmission tests were performed with the H.264/SVC stream [3] in order to analyze its capability of being transmitted over the Internet without losing its scalable properties. For that purpose, four different video sequences were encoded using the H.264/SVC encoder [7]. Two image definitions (424x240 and 848x480) were used, with four levels of temporal scalability (30, 15, 7.5 and 3.75 fps) and two levels of SNR quality.

TABLE I
Transmission Layers according to SVC Scalability Layers

Transmission Layer	**SVC NAL Unit**		
	D	***T***	***Q***
Layer 0	0	0, 1, 2 and 3	0
Layer 1	0	0	1
Layer 2	0	1	1
Layer 3	0	2	1
Layer 4	0	3	1
Layer 5	1	0, 1, 2 and 3	0
Layer 6	1	0	1
Layer 7	1	1	1
Layer 8	1	2	1
Layer 9	1	3	1

A. *Fragmentation and Reassembling of H.264/SVC Streams*

In order to explore the scalability options of the H.264/SVC bit stream, it must be fragmented in several transmission layers, taking into account the Network Adaptation Layer (NAL) unit types. To achieve this, the H.264/SVC bit stream is in a first stage partitioned into smaller files, called chunk files, which in turn were fragmented into several transmission layers, according to their NAL unit types and in response to the scalable spatial, temporal and SNR layers (i.e., (D,T,Q) identifiers). Table I represents the (D,T,Q) identifiers of each transmission layer and Figure 1 presents the Block diagram of the chunk and layer partitioning/reassembling functions.

The fragmentation process starts by analyzing H.264/SVC bit stream structures searching for NAL units of Supplemental Enhancement Information (SEI) type. At the encoding it was defined that each chunks starts by the SEI message. Afterwards, the splitting into transmission layers is based on the analysis of the tuple (D,T,Q) of the correspondent NAL unit.

Both processes for fragmentation and reassembling of the H.264/SVC bit stream were made in order to obtain a decodable sequence at the receiver, using a subset of the original quality and supporting real time H.264/SVC decoding, as supported by [8]. The real-time requirement also imposed that recently added receivers could be able to start decoding the video at any playback time; i.e., not necessarily at the beginning of a TV program. To this aim, we have imposed that any chunk should be independently decodable, setting the chunk length to 2 seconds. Different from traditional BitTorrent solutions and since the video was encoded in Variable Bit Rate, consecutive chunks normally have different sizes.

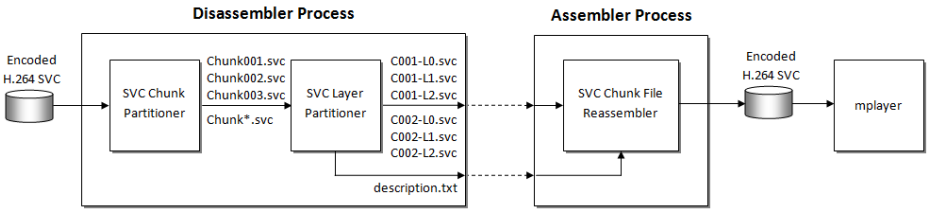

Figure 1. Block diagram representing the Fragmentation and reassembly processes of the H.264/SVC bit streams.

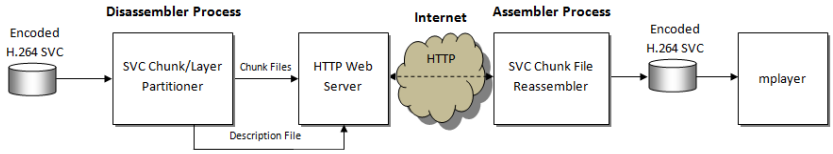

Figure 3. Block diagram representing the transmission of H.264/SVC bit streams using an HTTP Web Server.

During fragmentation of the original bit stream in chunks and layers an auxiliary description file is created containing the information of the interdependency of the SVC layers in each group of chunk files. This file was afterwards used as reference in the reassembling process.

Since some parts of the bit stream can be lost in the transmission, during the development of the reassembler process it was verified the need for an additional field that references the order of sequence of each NAL unit in the original H.264/SVC bit stream. A new 2-byte field called Sequence Number (SeqNum) was therefore added between the NAL units start code and the beginning of the NAL unit data [3], as shown in Figure 2.

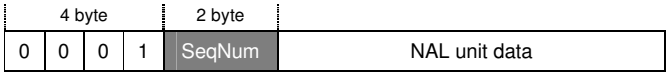

Figure 2. Example of the NAL unit structure with the 2-byte sequence number inserted field.

Finally, the reassembler module combines the chunk files, using the interdependency information of the H.264/SVC layers obtained from the description file and reorders the received NAL units using their sequence numbers. As the sequence number of the NAL unit field are only used for the multiplexing the NAL units at the reassembler process, they are discarded after this step.

B. Transmission of H.264/SVC Streams using a Web Server

In this step, a system capable of transmitting H.264/SVC bit streams using the HTTP protocol was implemented. To achieve this, minor improvements had to be made to the system described in section III-A. The description file coming out of the partitioning process was published online, together with the chunk and layer files of the H.264/SVC bit stream using an HTTP Web Server. Additionally the reassembler process was made to download these chunk files published in the Web Server.

The complete system, capable of transmitting H.264/SVC bit streams over the Internet is shown on Figure 3.

C. Transmission of H.264/SVC Streams using a P2P Network

From the architecture implemented in Sections III-A and III-B, the following step was to improve the system with the capability of transmitting H.264/SVC bit streams over the Internet using a P2P network. For that purpose, as explained in Section II, the BitTorrent P2P file-sharing protocol was chosen.

Given the system obtained in section III-B three relevant improvements needed to be made. First, the description file needed to be converted in a torrent file, as required by BitTorrent. For this, all the information of the description file was included in the new torrent file, together with the required URL of the tracker and the fixed piece size of the torrent. Figure 4, represents an example of such torrent file.

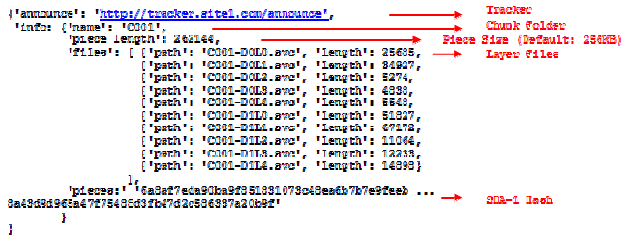

Figure 4. Example of a torrent file structure for a H.264/SVC bit stream.

The second required change made to BitTorrent was on the piece selection method. In general, BitTorrent protocol is not suited for real-time applications and a great part of this derives from its piece selection method. It does not respect the chronological order of the events in the bit stream when downloading the required pieces. To try to solve this issue, we tested a sequential piece selection method.

Finally, the third and last necessary improvement was to create a relationship between the torrent pieces over the network and the several SVC layers. For this, the chunk files

needed to be padded to match multiples of the fixed piece size defined in the torrent file.

Due to the excessive overhead caused by the padding of the chunk files, a 16 kByte fixed piece size was used (yielding aprox. 24% overhead) instead of the standard 256 kByte (aprox. 635% overhead) commonly used, as described in [5]. The choice of the 16 kByte fixed piece size was only limited by the minimum value defined by the software used in the tests.

The performed tests consisted in placing a seed online, with all the chunk files of the torrent file available through the execution of a common BitTorrent client. In the receiver side, the torrent file was given as input to the receiver process. As each peer registered in the swarm and after a few seconds the download of the pieces of the chunk files started in a sequential order. The receiver reassembled the H.264/SVC bit stream, which was then reproduced.

D. Analysis of the Implementation

The defined system was able to deliver the scalable video using a BitTorrent P2P overlay. However, in this stage several challenges were still unfulfilled. First of all, the torrent file, as it was specified, requires the content to be completely available before the startup of the transmission. Although it works properly with stored content, it does not support a real-time encoding, as required by this research.

Additionally, the piece selection process did not differentiate what layer and chunk should be requested first. In fact all layers and chunks had the same default priority, set by the BitTorrent protocol. Also, in a real time transmission of video there are two important elements that need to be supported: first, the delay between channel switching and the start of video reproduction needs to be reduced as much as possible; and content availability dictated that at least lower SVC layers should always be delivered in time to receivers for their reproduction. This last feature avoids the complete loss of certain chunks with the corresponding subjective quality degradation.

Until this point, all tests were based on practical experiments, using real software and computers connect to the Internet. Given the number of variables that can be used in such a P2P live distribution, which should be combined with a high number of peers, in the following steps we decided to implement a simulation framework using Matlab, in accordance with the BitTorrent mechanism and reflecting the system defined up to this point.

IV. SIMULATION FRAMEWORK FOR REAL-TIME H.264/SVC USING BITTORRENT

The simulation framework was implemented using a discrete event simulator developed using Matlab, in accordance with the BitTorrent protocol specification [5]. Among other features it included: the announce procedure between peers and the tracker of the swarm; the information exchange between peers regarding their different interests, the content availability and the requests for download of the pieces of the torrent files; together with the choke/unchoke mechanism of each peer in the swarm. For this environment to work as expected, an independent entity within a group of several pre-defined objects and variables was created in each peer, that translates: the upload and download available bandwidth; the neighbor peer list and the associated requests for pieces of each of these neighbor peers; the status of the choke/unchoke mechanism; and the entrance/exit instants in the swarm. During the execution of the simulation tests these variables were continuously changed and processed. The results obtained for each of these tests, reflect the sampling of the different values of these objects and variables for a set of simulations.

During this study the framework suffered several improvements and changes; however the compatibility of its behavior with the initial specification of BitTorrent was always taken into account. The result was an accurate simulation framework capable of a strict reproduction of the BitTorrent protocol behavior as described in [5]. This model enabled the adjustments of variables like chunk and layer priority and a faster measurement of the results, using a much higher number of terminals.

According to the characteristics of BitTorrent described in Section III, its adaptation to support a real-time transmission of H.264/SVC video, requires some improvements. In the following we describe these changes.

A. Incremental Torrent File Information

Since the BitTorrent protocol was developed with the purpose of file-sharing, the torrent file is a static text file containing the whole structure of the encoded file and the information required for downloading the content files. In a real-time encoding and transmission of video however, as new H.264/SVC chunk files are generated by the source peer, a mechanism is required to inform other peers within the swarm, about where these files are located.

To keep track of these updates, peers should have access to a simple content update mechanism, similar to the one available in an RSS feed. An RSS feed is a standardized technology to directly access to subscribed contents on websites, allowing users to access only new published content without having to manually inspecting all of the websites they are interested in. It works by requiring the access to a standardized XML file that the RSS reader software requests periodically, searching for updates.

In the following, the usage of an RSS feed was considered to support the periodic update of information about the availability of new chunk files among the peers of the swarm.

B. Sliding Window Piece Selection Method

Since the rarest-first piece selection method used by the BitTorrent protocol is not suited for real-time applications and a purely sequential piece selection method seems not to be the best option in terms of real-time, a new piece selection method was adopted.

A sliding window based solution as the one proposed in [9] seems to be more appropriate, as some layers could be dropped after a timeout, in case they are not received. Therefore a solution similar to the method described in [9] was chosen, with some improvements to support the scalability properties of an H.264/SVC bit stream.

Figure 5, shows an example of the sliding window selection method. The window contains the next N chunks/pieces of the different layers needed to reproduce the H.264/SVC bit stream in the original events order. Peers can only request pieces inside the sliding window and need to discard requests for pieces of events already reproduced.

When using different H.264/SVC layers, peers must decide which chunk and layer to request first. For this purpose a prioritization criteria was defined that combines both the SVC layer index and the remaining time to its reproduction. Based on this, lower layers, and chunks closer to playback, have been given higher priorities.

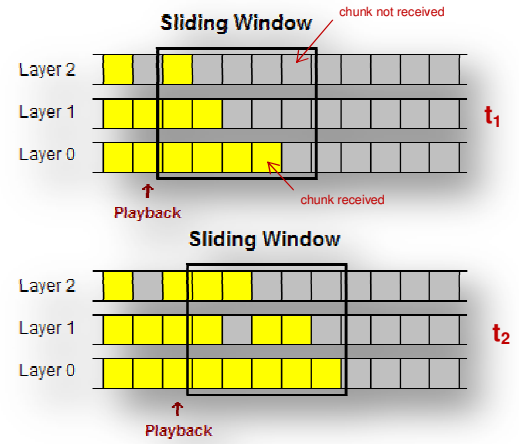

Figure 5. Example of a sliding window piece selection method for an SVC bit stream with 3 scalability layers.

In Figure 6, we can observe an example of such priority scheme. All chunks and layers outside the sliding window assume the normal value of priority of pieces in the BitTorrent protocol (priority=2), while priorities inside the sliding window increase according to the importance of the SVC layer (i.e. base layer is the most important layer) and decrease according to the chunk number.

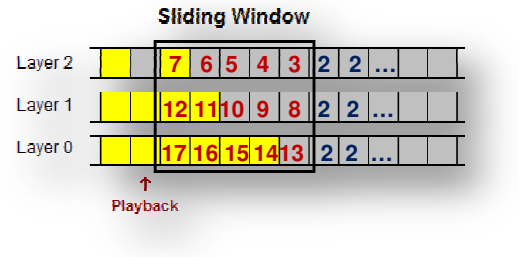

Figure 6. Example of the chunk prioritization criteria according to the layer scalability level and the time to download before being needed.

By either reducing or enlarging the window sizes peers can respectively request a higher number of chunks and layers, or concentrate in obtaining a lower number of chunks and layers.

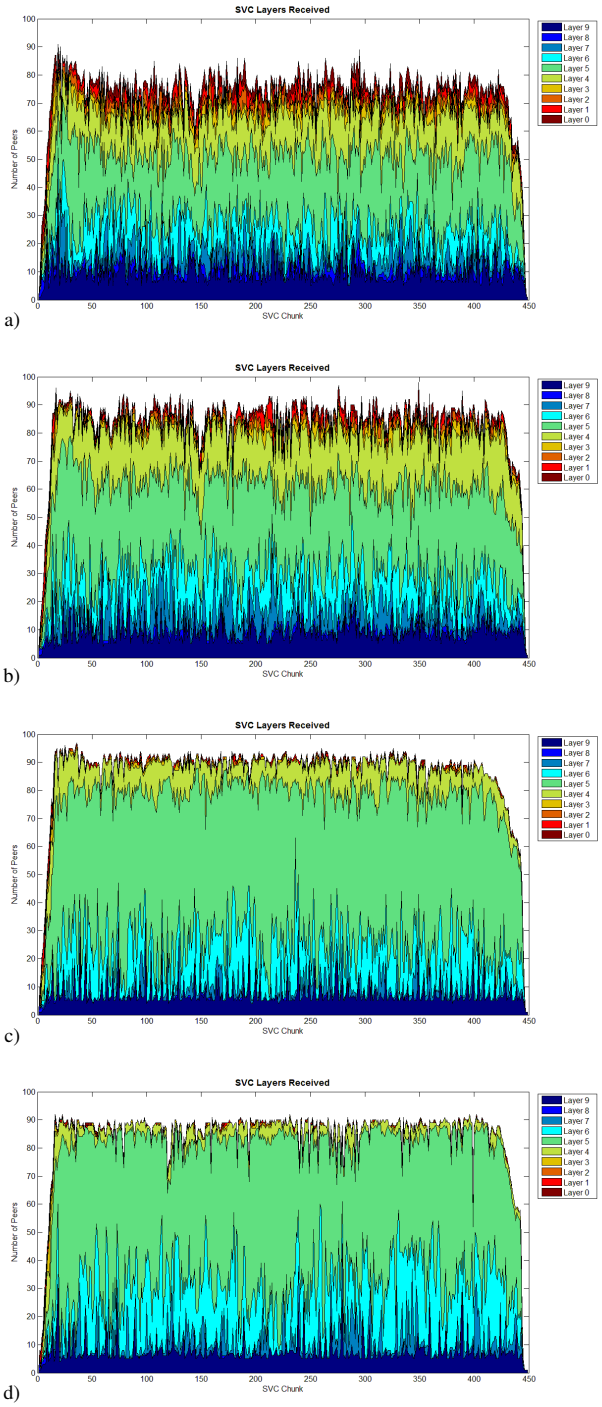

Figure 7. SVC layers received using different sliding window sizes: a) 2 chunks b) 3 chunks c) 4 chunks d) 5 chunks.

V. Simulation Results

The improvements referred in Section V were applied to the BitTorrent simulation model. Several tests were performed using a sequence of video encoded in H.264/SVC, fragmented in 10 transmission layers (as described in section V), with an

average bit rate of 1.45 Mbps. The tests considered a transmission on a swarm of 100 peers during 900 seconds. All peers had a maximum uplink rate of 2.0 Mbps and a maximum downlink rate of 20.0 Mbps. The maximum P2P upload rate of peers was limited to 90% of the uplink rate (i.e 1.8Mbps). During simulations all peers entered the swarm in a random distribution along the first 30 seconds of the experiment. The tests were made using four values for sliding window sizes, namely 2, 3, 4 and 5 chunks.

Figures 7 a), b) , c) and d) show the evolution in the number of layers received by the peers in the swarm as the chunks are being transmitted. It can be verified that the best results were achieved for window sizes equal to 3 and 4 chunks. The solution with 3 chunks guaranties that a higher number of receivers are able to get higher layers, with the drawback of a significant amount of peers not receiving any layers. On the contrary

Table 2 present the average number peers receiving each of the transmitted layers. For instance it can be verified that the number of peers receiving 6 layers increases by 31% when varying the sliding window size from 2 to 5 chunks and that the number of peers not receiving any layer decreases by 50%.

TABLE II
RATIO OF RECEIVED SVC LAYERS FOR DIFFERENT SLIDING WINDOW SIZES

	Sliding Window Size			
	2 Chunks	***3 Chunks***	***4 Chunks***	***5 Chunks***
None	22%	11%	9%	11%
Layer 0	78%	89%	91%	89%
Layer 1	76%	88%	91%	89%
Layer 2	73%	86%	91%	89%
Layer 3	70%	84%	90%	89%
Layer 4	67%	82%	89%	89%
Layer 5	54%	65%	80%	85%
Layer 6	29%	33%	26%	34%
Layer 7	19%	20%	11%	14%
Layer 8	13%	13%	8%	9%
Layer 9	9%	9%	7%	7%

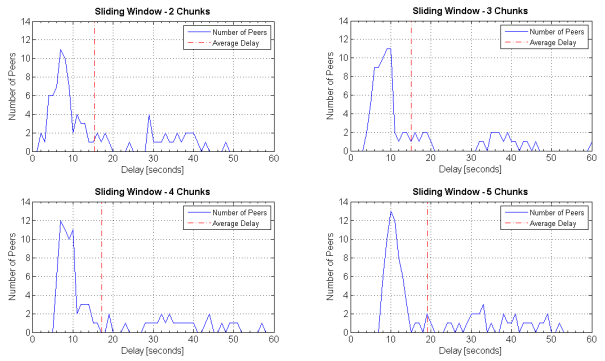

Figure 8. Histogram of delays verified between the original bit streams and content reproduction for each receiving peer and considering different sliding window sizes: 2, 3, 4 and 5 chunks.

Figure 8 presents the histograms of delays between the time of the original bit stream and its reproduction. The best result was achieved for a window size of 3 chunks, yielding a delay of nearly 15 seconds.

VI. CONCLUSIONS AND FUTURE WORK

This paper describes the implementation of a BitTorrent based H.264/SVC transmission, complemented by the implementation and testing of several features using a simulation model.

Globally, the results obtained have shown that the BitTorrent mechanism can be adapted to support a real-time H.264/SVC transmission over IP networks.

The tests performed using the simulation model have shown that a sliding window of 3 or 4 chunks was capable of delivering at least one layer to nearly all receivers. In terms of delay, between joining the swarm and the start of video reproduction, a window of 3 chunks has been capable of reducing it value to nearly 15 seconds. Both results show that, given the conditions defined for these tests, the adjustment of the window size influence the real-time behavior.

These results have also demonstrated that there are still several improvements that need to be performed to the system here described. Not only the number of receivers getting at least one layer should be increased but also the number of layers.

The need of a mechanism responsible for the reduction of the delay between the time of the original bit stream and the reproduction on the different peers should also be considered.

Regarding the increase of the number of SVC layers received and the maintenance of the BitTorrent swarm, alternative piece selection methods and the resilience to random entrance and exit of peers in the swarm should also be tested and analyzed in a future work.

REFERENCES

[1] Cisco Systems, Inc, “Cisco Visual Networking Index: Forecast and Methodology, 2010–2015”, White Paper, Jun. 2011.

[2] ITU-T and ISO/IEC JTC 1, ”Advanced video coding for generic audiovisual services, ITU-T Recom. H.264 and ISO/IEC 14496-10 (AVC),” Version 8, Jul. 2007.

[3] J. M. Monteiro, C. T. Calafate & M. S. Nunes, "Evaluation of the H.264 Scalable Video Coding in Error Prone IP Networks", IEEE Transactions on Broadcasting, vol. 54, no. 3, pp. 652-659, Sept. 2008.

[4] The Liquidculture Notebook, “The absolute majority of all Internet traffic is p2p file-sharing”, Internet: http://liquidculture.wordpress.com/2008/03/14/the-absolute-majority-of-all-internet-traffic-is-p2p-file-sharing/, accessed: Nov. 13, 2011.

[5] B. Cohen, “The BitTorrent Protocol Specification”, Internet: http://www.bittorrent.org/beps/bep_0003.html, accessed: Nov. 13, 2011.

[6] J. F. Buford, H. Yu, E. K. Lua, “P2P Networking and Applications”, Morgan Kaufmann, 2009.

[7] J. Vieron, M. Wien, and H. Schwarz, JSVM-9 Software, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Doc. JVT-V203, Jan. 2007.

[8] Sourceforge, “Open SVC Decoder”, Available: "http://sourceforge.net/projects/opensvcdecoder/", [Accessed Jun. 5, 2010].

[9] P. Shah, J. F. Pâris, “Peer-to-Peer Multimedia Streaming Using BitTorrent”, University of Houston, 2007.