PACKETIZING SCALABLE STREAMS IN HETEROGENUS PEER-TO-PEER NETWORKS

Alexandro Sentinelli, Tea Anselmo, Pasqualina Fragneto,
Amit Kumar, Beatrice Rossi

STMicroelectronics, via Olivetti 2, Agrate Brianza (MB), Italy
{tea.anselmo, pasqualina.fragneto, amit-agr.kumar, beatrice.rossi, alexandro.sentinelli}@st.com

ABSTRACT

After the extensive effort dedicated by both Academia and Industry in the area of peer-to-peer (P2P) streaming by looking at models and network algorithms to achieve optimal load distribution, recent works are moving forward to enhance the overall P2P systems efficiency by focusing on specific video codecs and packetization techniques at application layer. The purpose to integrate different technologies with the aim to design full streaming solutions requires a joint efforts from the market and EU projects community. Many engineering issues have been brought out in terms of compatibility and integration that need to be addressed. In this paper we focused on a bottleneck discovered during the packetization step between the Scalable Video Coding (SVC) streaming module (during content creation) and the P2P engine before delivering the packets over the network. We compared the a priori fixed packet size solution adopted in P2P-Next project with a codec aware approach, gaining performance in terms of network overhead. We performed experiments with various sequences aiming at overall P2P streaming efficiency and measured the bandwidth cost under various conditions. Since we focused on the application layer, our statistics analysis can be helpful in designing P2P streaming solutions dealing with any network type.

Index Terms— SVC, Content creation, Video Coding, P2P, Bitrate Control, Network Overhead, Live streaming

1. INTRODUCTION

Despite the Global Economic Slowdown, a recent IPTV forecast projects 5.5 million subscriptions by 2013 in the US alone. In Europe, according to [11], the number of IPTV subscriptions will increase by 92%, from 15.4 million in 2009 to 29.6 million in 2015, making it the fastest-growing among all pay-TV platforms. The technological challenge is to satisfy various users with different network and devices all accessing to same content. Nowadays several providers offer the same content at different qualities (bit-rates) with the aim to satisfy users with different devices or available network bandwidth. Unfortunately, the network efficiency does not benefit as well because the system treats the same content like two different bitstreams that are independently encoded and transmitted over the network. The Scalable Video Coding (SVC) standard, instead, plays upon the concept of layered video coding with one bitstream multiplexing many inter-dependent substreams, formed by a base layer with minimum quality (common to every peer) and a number of enhancement layers to increase the quality for more exigent users. By moving to P2P we can improve the overall network efficiency of the distribution. The idea of combining together SVC and P2P gathered a lot of interest from the industry that pushed the R&D community step-by-step toward the in depth research or realization of SVC or P2P technology individually or altogether for streaming solutions. Different technologies have to be embodied and a number of problems are being faced in terms of integration and compatibility which may cause sometimes significant drop in performance. In this work we want to point out issue(s) that came out while integrating the SVC streaming module (for content creation) along with P2P engine. We need, in fact, an interface that receives the SVC stream and maps it into input packets for the P2P engine. Since one of the main constraints is the backward compatibility with existing P2P clients using metadata file like torrents, thus there is a need to maintain a constant block or piece size, the encapsulation or packetization step costs some bandwidth to the system in terms of overhead introduced into the network. In this work we compared two types of packetization methods and measured their performances in terms of overhead. The first solution has been developed in the European project P2P-Next. In this paper we will be using the term "block", although some existing P2P clients uses term "piece" for the same. The algorithm basically takes a group of N_{fr} frames from the SVC stream coded at constant bit-rate and encapsulates them into one fixed size block ready for the P2P engine. In our algorithm we encapsulate the same number of pictures N_{fr}, equal to the Instantaneous Decoding Refresh (IDR) period, into a variable number of fixed size smaller blocks as compared to fixed number of fixed size block chosen in P2P-Next. The IDR coded picture must be inserted at the beginning of a new block and entirely contained in it as it serves as random access point in case of lost packets and is uniquely decodable (independent on previous or next block within same layer). A frame, other than IDR, can be eventually cut and fit into two adjacent blocks, but the blocks size remains same, thus the compatibility with torrent clients is maintained. Intuitively, when the size of N_{fr} coded pictures exceeds the target constant rate due to the variable complexity of video content, the use of a variable number of blocks allows for packetizing all the frames. The paper is organized as follows. After the related works and a brief description of the overall basic architecture from the P2P-Next, we will focus on encoder chain starting from content creation (encoding raw media stream with a Constant Bit-Rate control algorithm) until the P2P block-encapsulation stage, the core of our analysis. Here we compare the two P2P packetization methods using publicly available video sequences verifying also the expected qualitative behavior with a mathematical model. The performance of the two solutions is measured in terms of overhead introduced into the network. Finally we provide the conclusions.

2. RELATED WORKS

As layered coding feature of SVC can fit perfectly well with the need of heterogeneous integrated networks, bandwidth fluctuations at the physical layer side offer many research opportunities for cross-layer optimization. In [1] the authors analyze problems related to delay, network connection errors, and the potential use of SVC standard in variable network environments while in [2] the authors discuss the use of multi-layered/multi-view content coding techniques for streaming, adaptation for various kinds of networks, e.g. xDSL, WiMAX. Some open issues have been widely discussed in [3] about multimedia streaming architecture using single or multiple sources of content distribution. The closest work related to our analysis is the P2P system proposed by Capovilla et al. [8]. In Fig. 1 the producer-side architecture describes all steps from SVC encoding till packet ingestion into the core of the P2P engine (NextShare Core). The topics addressed include the encoding process, splitting of the bitstream, creating metadata for initializing RTP session based on the bitstream's supplemental enhancement information (SEI), sequence parameter sets (SPS) and picture parameter set (PPS), the bitstream packetization, and finally the ingestion into the P2P engine. The consumer-site architecture describes all steps from receiving the bitstream through P2P engine till decoding/rendering the bitstream with an SVC media player. Our investigation focuses on the upper part of the producer side (Fig. 1 - dashed line). We point out a remark about the layer definition, as it is different from the SVC standard [5]. The temporal scalability is not considered and the layers are interpreted as generic video files as in the NextShare (P2P-Next) specification [8].

3. CHAIN DESCRIPTION

The chain (see Fig. 2) consists of an SVC encoder featuring CBR (constant bit-rate) capability, which gives an SVC elementary stream with constant bit-rate, which is further passed to the splitter (NALU Demux): the output is a set of files (or substreams), one per layer.

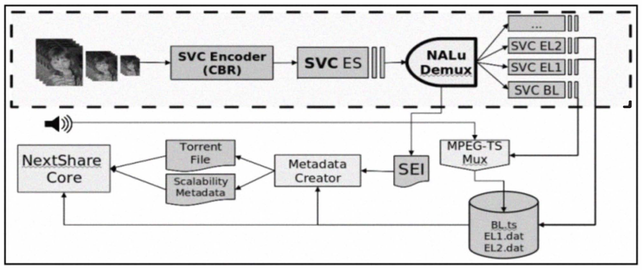

Fig. 1. P2P-Next project: the NextShare streaming chain.

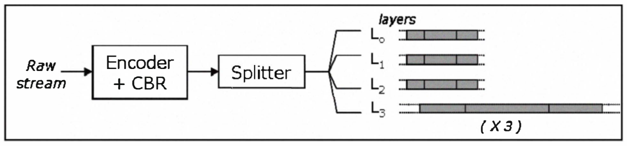

Fig. 2. Compact view of our chain.

In the next sections we will explain the CBR algorithm and the two splitters we are comparing, using SVC encoded video within the context of P2P streaming.

3.1. Integrating SVC in P2P networks

Scalable Video Coding is an extension of the H.264/AVC [4] and it is one of the latest video coding standards developed by the Joint Video Team (JVT) of the ITU-T Visual Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). Since principles of scalable coding are widely known to the video coding community and an in-depth description of SVC in not the aim of this article, readers are referred to [5] for a more detailed overview of the standard, while an analysis of coding efficiency is reported in [6]. Here below we focus on the properties of SVC that make this standard easily adaptable to heterogeneous networks and especially efficient in P2P environments. SVC provides scalable video streams, which are composed of a base layer and one or more enhancement layers: each enhancement layer can improve the temporal rate, the spatial resolution, supporting both dyadic and arbitrary resolution ratios, and the quality of the video content. Scalable bitstreams offer easy adaptation to varying network conditions and terminal capabilities: the scalability property refers to the capability of adapting the bitstream to varying terminal capabilities, network conditions and end user preferences by selectively discarding parts of the scalable bitstream and still obtaining an SVC decodable bitstream. This ease of adaptability brings more flexibility that becomes important to P2P systems in case of high churn rate (when peers change channels or disconnect) or sudden network congestions, which may affect dramatically the user experience. That is why many P2P systems use to relax playback delay constraints and set large buffers to overcome changes in the overlay network. Moreover, in terms of network efficiency, an overall improvement is shown in Fig. 3.

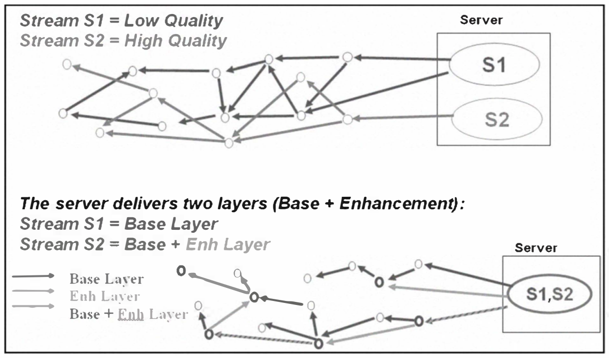

Fig. 3. P2P networks: independent video encoding (top) vs. SVC coding (bottom).

In most of the existing P2P streaming solutions the interesting media contents are usually delivered at two independent network overlays giving the possibility to the users to choose the quality/resolution they want. These bitstreams are independently encoded, so the network overlay is made by two independent sub-overlays.
Each end-user belongs to one of the sub-overlays and consumes the associated bitstream (Fig. 3, top).

Although this solution meets the user requirements it does not exploit completely the benefits of P2P systems. In fact, by using SVC (Fig. 3, bottom) the two sub-overlays are actually a single overlay embracing the entire peer population that shares the base layer and a sub-set of peers delivering only the enhancement layer. The availability of peers is maximum in numbers for base layer which is always common to the whole overlay. If we want to exploit all the advantages with this synergy (SVC/P2P) the P2P client needs an interface which receives the video stream and fits it into packets, all of same size, also avoiding the backward compatibility issues with nowadays existing P2P streaming solutions with fixed packet size. In order to build video packets referring to different qualities and same block size, we need to act at the beginning of the chain by properly modeling/organizing the encoded bitstream.

The encoded bitstream is formed by a series of data packets called NALU (Network Abstraction Layer Unit), each containing either an entire encoded image, or part of it, or the header information needed to properly decode the bitstream (i.e. Sequence and Picture Parameter Sets, Scalable and Supplemental Enhancement Information). In SVC, all representations of the same image with different layer identifier (spatial or quality) for a time instant form an Access Unit (AU). In order to map SVC units (NALUs) into input packets for the P2P engine, a rate control is required to adapt the intrinsic variability of bitstream rate to the packetization process. Constant Bit-Rate (CBR) algorithm acts on the QP (Quantization Parameter) value in order to adapt the variable rate bitstream to a limited bandwidth channel. The results shown in this paper refer to the buffer-based CBR control method proposed in [7], which is based on buffer management to avoid overflow and underflow events, and is suitable for multiple layer coding: the algorithm tries to achieve, at the end of each Intra period, the same buffer fullness that was before encoding the last Intra picture. As a consequence, it performs a constant bit-rate encoding since every Intra period of length *IDR* consists of about the same number of bits:

$$\textit{Target IDR size} = \textit{AvgBitPict} \cdot \textit{IDR} \qquad (1)$$

where *AvgBitPict* is the average amount of bits per picture obtained as the ratio of the target bit-rate and the sequence frame rate.

Table 1. Simulation results of SVC encoder with CBR.

Sequences	Layer 0		Layer 1		Layer 2		Layer 3	
	BR [kb/s]	Err [%]	BR [kb/s]	Err [%]	BR [kb/s]	Err [%]	BR [kb/s]	Err [%]
City	395.28	-1.18	798.10	-0.24	1,191.17	-0.74	2,400.30	0.01
Crew	395.28	-1.18	794.52	-0.68	1,195.82	-0.35	2,376.83	-0.97
Soccer	395.26	-1.18	800.24	0.03	1,186.58	-1.12	2,383.78	-0.68
Flight	400.97	0.24	800.29	0.04	1,203.26	0.27	2,405.03	0.21
Tree	408.01	2.00	803.57	0.45	1,219.28	1.61	2,412.29	0.51
Home	400.06	0.01	799.97	0.00	1,201.22	0.15	2,404.22	0.18

In order to produce a nearly constant amount of bits *Target IDR size*, the algorithm checks the buffer fullness and compares the current buffer occupancy level with the target level: in case of emptiness detection, the QP is increased, while in case of possible overflow, the QP is decreased and filler NALUs are inserted. A more detailed description of the CBR algorithm can be found in [7]. The encoder along with buffer based CBR is able to achieve good bit-rate control performance and, at the same time, to maintain fine uniform image quality throughout the sequence. Table 1 summarizes the final bit-rate error obtained with JSVM reference software [9] and the buffer-based CBR for the scalable configuration including a base layer and a spatial enhancement layer (Layer 0, Layer 2) with two quality levels (Layer 1, Layer 3).

The ITU-T test sequences Crew, City, Soccer are 300 frames long and they have been used with CIF to 4CIF spatial resolutions at 30 Hz. The sequences Home, Flight and Tree [10] are 9000, 1200 and 600 frames long respectively, and they have been coded with QVGA to VGA resolutions, Home at 24 Hz while the last two at 25 Hz. The tested coding parameters are the ones referred in Table 1. The target bit-rates for the four layers are respectively 400 kbit/s, 800 kbit/s, 1.2 Mbit/s and 2.4 Mbit/s. For all our simulations the IDR period is equal to 64 frames.

The CBR working at IDR target size comes in handy for the need of having an IDR at the beginning of a new block.

Therefore, when NALUs are provided by SVC encoder to the P2P engine, they are packetized into single or set of blocks, encapsulating an IDR period to be provided to P2P engine and further shared over P2P network. In our proposed solution each block contains only the data belonging to a unique layer (among base and enhancement layer(s)) of the scalable bitstream. In order to extract a specific quality from the bitstream it is sufficient to map blocks into the main bitstream.

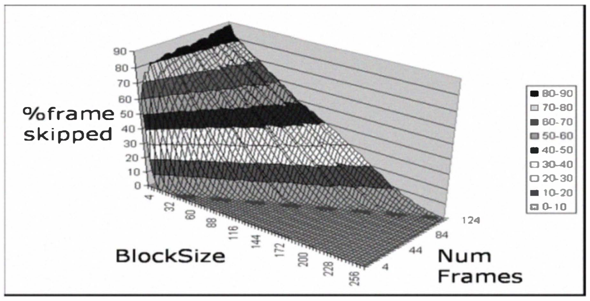

Fig. 4. Skipped (%) frames f(B_s, N_{fr}).

Such mapping needs a mask to indicate the number of blocks per layer and a priority scheme to specify how to insert blocks into the stream for synchronization. The synchronization mapping scheme can be defined in two ways. The first one declares a number of blocks per IDR per layer that is kept invariant during the streaming. For example a mask [1,1,1,3], means that every layer needs one block per IDR except the highest one (3 blocks). In the second approach we are aware of each IDR, thus leading to declare a variable number of blocks per layer all along the stream. In Fig. 5 we show an example with a stream containing 3 layers (Base + 2 Enhancement layer).

3.2. Splitter module

The splitter basically works in the same way as a demuxer. As the SVC elementary stream encapsulates all the layers into one file, the splitter application creates a separate file per layer and fits the corresponding NALUs into it. Each file can be seen as a continuous stream of blocks belonging to the same layer until the content's end. In the following paragraphs we will describe the two splitters, P2P-Next Splitter and Adaptive splitter.

P2P-Next Splitter. The input parameters provided to P2P-Next splitter are N_{fr} ("Number of Frames per Block") and B_s ("BlockSize"). When a NALU doesn't fit into the block it is simply dropped. Therefore, when receiving the first frame of the next block, layers are always synchronized among base and different enhancement layer(s). If the block advances some space (bits) after containing N_{fr} frames, a filler NALU is added in order to maintain constant block size.

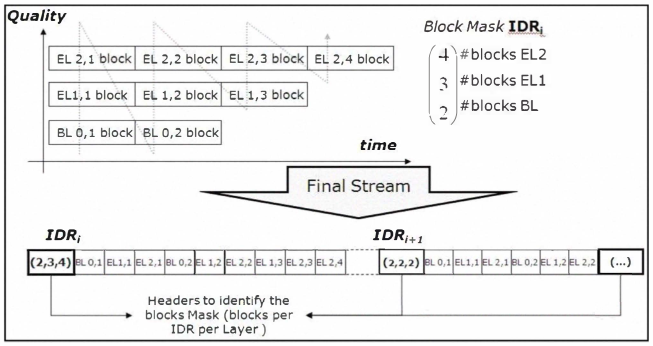

Fig. 5. General synchronization priority scheme.

The rate of frames skipped during the streaming depends highly on the input variable couple (B_s, N_{fr}). In general, we can say that if we increase N_{fr} with respect to B_s, we will have a higher number of frames skipped though less filler NALU per block. On the other hand, if we increase B_s with respect to N_{fr}, we will output sub-bitstreams with a higher percentage of filler NALU while decreasing significantly the percentage of frames skipped. A trade off is obtained by considering the average size of one frame in a stream with respect to a defined target bit-rate, and to maintain this ratio while choosing the parameters N_{fr} and B_s. In particular, our experiment uses an bitstream encoded with SVC encoder featuring CBR with encoding parameter as 400 Kbit/sec, 25 fps (frame per seconds), which means each frame on average is 2 kB in size to keep the synchronization with any streaming application designed over our proposed algorithm. We have also analyzed the effects of the relationship between [B_s, N_{fr}] and the percentage of frames skipped in Fig. 4. As per our proposed algorithm required to maintain the ratio given by the chosen target bit-rate, it is evident that the ratio between N_{fr} and B_s must be 1:2. The dashed line represents the optimal trade-off between wasted space and the percentage of skipped frames, and it continues following the relationship B_s=2*N_{fr}. Considering a practical P2P application design perspective with N_{fr} = 64 and frame-rate 25 fps, in order to keep all frames in the single block it is suggested to choose a target bit-rate a little bit lower than 400 kbps, or B_s little bit larger than 128 kB. As the rate precision of the CBR algorithm may depend on video content complexity, choosing target bit-rate or B_s in this way will avoid any frame skipping during splitting operation.

In actual *splitter*'s implementation each block contains a group of 64 frames (IDR period) starting from an IDR picture. If the IDR size exceeds B_s, the future incoming NALUs (for same IDR period) are discarded. While, in the other case, if the IDR size is less than B_s then the size corresponding to (B_s less IDR period size) is completed with a filler NALU.

Adaptive Splitter. In this version, the splitter receives only one parameter as input: the block size, which is relatively smaller in size with respect to the previous version of splitter discussed above. The main feature of this approach is the possibility to cut the IDR period into multiple blocks. This feature is suitable to adapt the amount of information to the content of the stream itself, or to change/choose the quality on the fly. In case of fairly static video scenes lasting for few seconds, the encoder needs less bits to encode it with respect to a scene with many objects moving around; therefore, depending on the specific video scene, the number of blocks per layer can be adapted to the amount of information of each IDR. It is evident that smaller the B_s, smaller the P2P protocol overhead in the whole network will be. On the other hand, reducing the block size very small will proportionally increase the communication overhead as compared to exchanged blocks among peers.

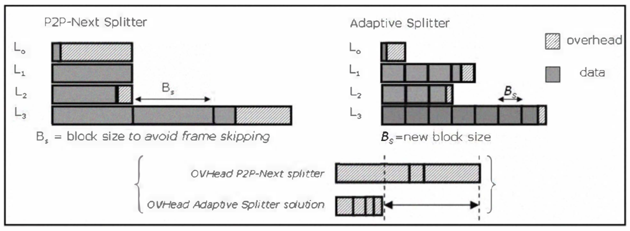

Fig. 6. Content size-aware solution. The Adaptive Splitter allows cutting IDR of any layer into a variable number of blocks.

In our experiment we have chosen a range of block size starting from 16 till 512KB, considering the block size used by various BitTorrent-protocol based applications. As shown in Fig. 6 the network overhead decreases remarkably. Note that the mask [1,1,1,3] used in the first version of the Splitter, may suggest that the IDR of the highest layer is cut into 3 blocks: once chosen the target bitrates, the contribution of the highest layer is 3 times the contribution of each lower layer, no matter what is the size of the data. Instead our proposed solution adapts dynamically the number of required blocks per layer for every streamed IDR period, as shown in Fig. 6.

Moreover, we remark that, if we want to be sure that no frames are dropped during the streaming session, the P2P-Next splitter needs to have a priori knowledge of the maximum IDR period size (among all the IDR periods) in the entire bitstream. That may result in high buffering or wasted bandwidth for the smaller IDR. The Adaptive Splitter instead doesn't require prior information about maximum size of the IDR and eventually is more suitable to be used in live streaming scenario.

4. EXPERIMENTS AND RESULTS

In this section we have considered six different video sequences (see Table 2) and compared the performances of the two proposed splitters. First, we will analyze the P2P engine by computing the percentage of frames skipped and the wasted space as result of P2P-Next splitter for different B_s values. In order to avoid frames skipped situation, we have performed several simulations with increasing B_s until the number of frames skipped reaches zero for each IDR period.

Table 2. P2P-Next splitter vs Adaptive Splitter.

Sequences	frmR [fr/s]	P2P Next Splitter		Adaptive Splitter			Gained OH [%]	Gained BW [%]
		MinSize [kB]	OH [%]	Range [kB]	Min OH [%]	Avg Min OH [%]		
City	30	119	19%	[16-119]	5.4	13.5	71	11.4
Crew	30	132	29%	[16-132]	6.5	15	77	17.4
Soccer	30	126	26%	[16-126]	6.3	14.3	67	15.6
Tree	25	165	30%	[16-165]	5.7	13.6	81	18.6
Flight	25	138	14%	[16-138]	4	8.5	77	11.9
Home	24	240	38%	[16-240]	5.7	13.1	85	23

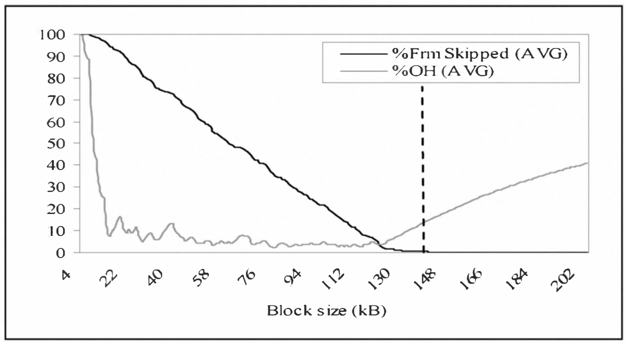

Fig. 7. P2P-Next Splitter Performances ("Flight" sequence).

For the sequence "Flight" (see Fig. 7) the minimum B_s to avoid frames skipped situation for each layer is 138 kB. This leads to an overhead (measured in percentage of wasted space over the output bitstream) of 18% for each layer. Note that only a block size bigger than 138 kB will guarantee the sequence to be encoded without any frame skipped. Then we analyze the corresponding results obtained by using the Adaptive version of the splitter that, as expected, doesn't skip any frame. Theoretical behavior of the average overhead per IDR period as function of B_s has been modeled as follows: in an ideal situation, when operating with a bit level precision CBR, theoretical overhead can be computed as the difference between B_s and the integer remainder of Target IDR size divided by B_s. Percentage of average overhead per IDR period for a fixed B_s becomes:

$$\mathrm{OH} = \frac{B_s - (\text{Target IDR size} \pmod{B_s})}{\lceil (\text{Target IDR size}/\mathrm{B_s}) \rceil \cdot \mathrm{B_s}} \cdot 100\% \qquad (2)$$

Results are shown in Fig. 8. When the overhead decreases, the IDR size gets close to a multiple of B_s. Moreover, peaks disappear when B_s becomes greater than the target IDR size (in this case 128 kB). Percentage of average overhead per IDR period as function of B_s is shown in Fig. 9. Trend meets the theoretical behavior with fluctuations determined by the CBR. Comparing results obtained with the two versions of splitter for the sequence "Flight", we observe that behaving in a different way as compared to P2P-Next splitter, the

Adaptive version allows to consider B_s values within the range of 16 kB to 138 kB while having no frames skipped.

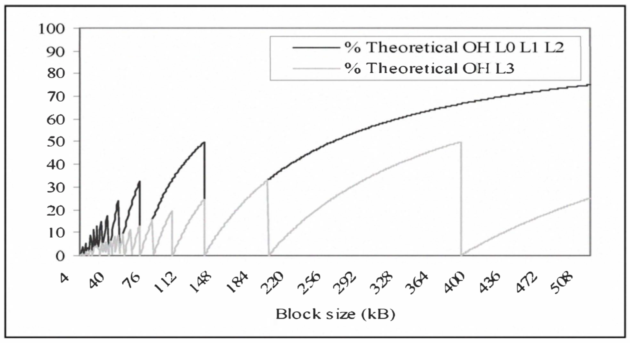

Fig. 8. Theoretical behavior of Adaptive Splitter.

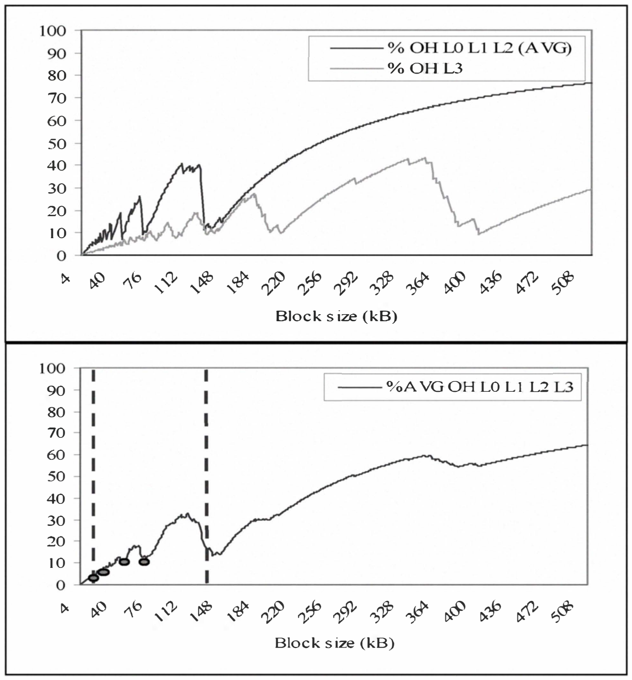

Fig. 9. Adaptive Splitter performances ("Flight" sequence).

In such range we find some minimal overheads of respectively 4%, 8%, 12.5% and 13%. Region contained by dashed lines describe the range within which Adaptive Splitter can outperform P2P-Next Splitter. On the whole, the adaptive approach gives up to a 77% decrease in overhead (see Table 2) as compared to P2P-Next splitter, which saves bandwidth cost up to 11.9%.

Results for remaining sequences follow the same trend as explained above (see Table 2). We also performed our experiment framework on a considerably long sequence (9000 frames ~ 6 minutes), extracted from the movie ("Home – Earth") (see Table 2). Finally, observe that, in order to obtain this performance gain of the overall system, there is a small overhead introduced which makes the splitter dynamically "adaptive". For each IDR, in fact, we have to add the number of blocks per layer, the temporal position and the layer index itself (Fig. 5). If we consider a 2 hour movie or live streaming at a bit-rate of 400 kbits/s with a SVC sequence splitted in four layers, thus adding for each IDR 2 Bytes for the temporal indexing (if fps = 25 and IDR period = 64, 2 Bytes may address more than 30 hours of streaming), 1 Byte for layer indexing and 1 Byte for the number of blocks per IDR/layer, we obtain an additional cost of only 11 kB per layer. Therefore, an overhead of only 44 kB for two hours of movie streaming with its highest quality is sufficient to save easily ~ 20% of the bandwidth.

5. CONCLUSIONS

We have described the architecture of an P2P based streaming solution using SVC encoding featuring CBR and interfacing a P2P engine from the raw video file till delivery of specific video layers in a BitTorrent fashion. We have compared performance in terms of overhead of the whole chain (encoder + splitter) while using the P2P-Next Splitter interface and our proposed solution of Adaptive Splitter that is able to extract the size of each IDR during the streaming. Results show a remarkable gain has been obtained using the Adaptive Splitter interface (confirmed by our mathematical model). Such an Adaptive Splitter is also a good candidate for live streaming scenario because it does not require the knowledge in *a priori* of block size (large enough) to preserve all frames of each IDR period: it just adapts the size for each IDR during the session.

6. REFERENCES

1. A.Z. Spector, "Achieving application requirements", *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, pp. 19-33, 1989.
2. T. Zahariadis, O. Negru and F. Alvarez. "Scalable content delivery over P2P convergent networks", IEEE ISCE'08, 2008.
3. M. Djamal-Eddine and M. Mubasher, "Open issues in P2P multimedia streaming", MULTICOMM2006, 2006.
4. ITU-T Rec.H.264 and ISO/IEC 14496-10. Advanced video coding for generic audio-visual services, March 2010.
5. H.Schwarz, D.Marpe and T.Wiegand, "Overview of Scalable Video Coding Extension of the H.264/AVC Standard", *IEEE Trans. Cir. Syst. Video Tech.*, 17(9), pp. 1103-1120, 2007.
6. M.Wien, H.Schwarz and T.Oelbaum, "Performance Analysis of SVC", *IEEE Trans. Cir. Syst. Video Tech.*, 17(9), pp. 1194-1203, 2007.
7. T.Anselmo and D.Alfonso, "Buffer-based Constant Bit-Rate Control for Scalable Video Coding", *PCS 2007*, 2007.
8. N. Capovilla; L. Mapelli; A. Kumar; A. Bakker; R. Petrocco; M. Eberhard; M. Uitto; "NextShare intermediate integration", d216217 P2P-Next, deliverable 6.5.3, June 2009.
9. Joint Scalable Video Model, version 9.15, available from: pserver:jvtuser@ garcon.ient.rwth-*aachen*.de:/*cvs*/jvt
10. http://media.xiph.org/video/derf/
11. http://www.analysysmason.com/About-Us/News/Insight/European-IPTV-subscriptions-to-double-in-five-years/