SVEF: an Open-Source Experimental Evaluation Framework for H.264 Scalable Video Streaming

Andrea Detti, Giuseppe Bianchi, Claudio Pisa,
Francesco Saverio Proto, Pierpaolo Loreti
University of Rome Tor Vergata
{andrea.detti, claudio.pisa}@uniroma2.it

Wolfgang Kellerer, Srisakul Thakolsri, Joerg Widmer
DoCoMo Euro Labs
Munich, Germany
{kellerer,thakolsri,widmer}@docomolab-euro.com

***Abstract*—This paper describes the H.264 Scalable Video coding streaming Evaluation Framework (SVEF). This is the first open-source framework for experimental assessment of H.264 Scalable Video Coding (SVC) delivery over real networks. Effectively adapting of the transport of an H.264 SVC stream to time-varying, bandwidth constrained, and loss prone networks is an important research area. However, very little experimental work has been performed due to the unavailability of real-time H.264 SVC players, the limitations of existing decoding software libraries when challenged with network-imparied received SVC streams (e.g., affected by random loss of Network Abstraction Layer Units – NALUs), and the lack of solutions for SVC streaming support. SVEF overcomes these issues by developing missing components and by integrating them in a hybrid online/offline experimental framework. We believe SVEF will be of significant help to the research community interested in experimentally benchmarking their own proposed SVC adaptation approaches and delivery mechanisms. As a proof-of-concept of SVEF, we provide the experimental performance evaluation of an SVC *cross-layer in-network scheduler* in a Wireless LAN hot spot scenario.**

I. Introduction

Wireless networks are usually characterized by a network capacity that may significantly vary over time. Such bandwidth fluctuations may be caused by the arrival and departure of traffic sessions competing for access to the shared medium, as in the case of a Wireless LAN [1]. Moreover, channel quality variations may trigger physical rate adaptation mechanisms [2], which cause frequent and step-wise abrupt changes of the available capacity. Most importantly, random loss of packets is frequently due to network congestion and wireless channel impairments [3], and further affects the available rate.

Scalable Video Coding (SVC) is a very promising encoding technique that allows to adapt to such variable bandwidth conditions [4]. Its basic concepts have been investigated by the research community for almost two decades, and its exploitation sped up by the recent (2007) finalization of an SVC specification in the framework of the ITU H.264 advanced video coding standards family [5].

An SVC stream is composed of multiple "layers" which carry incremental video enhancement information. More specifically, H.264 SVC introduces three enhancement dimensions: video frame size, video frame rate, and video quality. In SVC, adaptation of the video stream to bandwidth constraints or bandwidth fluctuations is performed by dropping (partly or entirely) one or more enhancement layers. This avoids run-time video transcoding, which has a significant processing overhead expecially for high quality video. H.264 SVC provides a very smooth adaptation that be performed at a granularity level as low as that of a single Network Abstraction Layer Unit (NALU). Such adaptation not only requires information about the available channel bandwidth, but it must also be done intelligently. For instance, dropping NALUs belonging to a lower layer makes all the received corresponding higher layer NALUs useless.

Many SVC adaptation solutions have been proposed [6], [7], [8], [9], [10]. They are either locally implemented at the video server, using bandwidth estimates obtained through feedback inherent in the transport protocol or some other control protocol, or implemented within the network, e.g., in *middleboxes* such as intermediate proxies, or in a wireless base station, where capacity information is available. This also allows for *cross-layer* scheduling to improve the adaptation efficiency.

Despite the large interest in H.264 SVC adaptation and numerous proposed approaches, to date little experimental work has been carried out on real network testbeds, especially when random NALU losses may be encountered. We believe the reasons for this are twofold.

The first relates to the fact that JSVM [11], the existing reference open source software for H.264 SVC coding/decoding released and maintained by the MPEG/ITU Joint Video Team, in its current version (9.15) is not able to decode video streams affected by out of order, corrupted, or missing NALUs. However, these issues frequently occur when transmitting SVC streams over unreliable wireless channels.

The second reason is the lack of freely available H.264 SVC streaming servers as well as clients (solutions such as those provided in [12] are restricted). For the case of servers, it is worth to remark that, to date, the support of H.264 SVC over the widely used Real-time Transmission Protocol (RTP) is still work in progress [13]. This may be a reason why, to the best of our knowledge, no public domain software appears available.

These shortcomings have motivated us to develop, and publicly release an open-source software framework SVEF [14], aimed at the performance evaluation of H.264 scalable video streaming. SVEF uses JSVM for H.264 SVC coding/decoding, but includes additional software to i) support H.264 SVC

978-1-4244-4671-1/09/$25.00 ©2009 IEEE

video streaming over IP, through encapsulation of NALUs in a simplified RTP structure, and ii) support receiver side decoding and reproduction of an SVC stream affected by arbitrary NALU losses and playout delay constraints. The latter is done in an offline fashion, over a raw NALU trace received at the client. The same uncompressed video that would have been displayed by a real video player client at the user side is, in SVEF, obtained through post-processing of the raw NALU trace, and subsequent application of i) a *Filter* for NALU decoding dependency and playout delay checking, ii) a custom method to extract and decode the resulting video stream, and iii) an offline *Filler* devised to provide a simple form of error concealment.

An important part of this paper is dedicated to the application of the SVEF evaluation framework in a proof-of-concept scenario of SVC delivery over a Wireless LAN hotspot [1]. In addition to showing the benefits provided by a *cross-layer NALU scheduler* running in a middlebox placed at the boundary of the WLAN domain, such an example scenario allows us to i) describe supplementary techniques employed in SVEF for solving practical problems such as fragmentation of large NALUs, and ii) present an example performance assessment. This performance evaluation is based on the metrics of luminance PSNR (Peak Signal-to-Noise Ratio) and number of lost video frames with respect to the unencoded video file, as well as a novel performance metric called *transmission efficiency*, devised to quantify the usefulness of the packet transmission process.

II. H.264 SVC Background

An H.264 SVC stream is defined as a sequence of NALUs. A NALU is composed of a header and a payload carrying, partially or entirely, an encoded video frame. The NALU header contains information about the type of data and its relevance in the decoding process. From the information reported in the NALU header (for full details see [5], [6]), we are specifically interested in three parameters: dependency_id (DID), temporal_id (TID), and quality_id (QID). Each parameter determines a specific scalability feature. DID allows *Coarse Grain Scalability*, TID allows *Temporal Scalability*, and QID allows *Medium Grain Scalability*.

Coarse Grain Scalability (CGS) provides the ability to *coarsely* adapt video properties, e.g., the video's spatial resolution from CIF to 4CIF. The video should be encoded with a suitable set of coarse enhancement sub-streams, called *dependency-layers*. The DID parameter identifies the dependency-layer to which a NALU belongs. The decoding of a NALU having $did > 0$ depends on NALUs belonging to the dependency-layer $did - 1$, and having the same value for the TID and QID parameters. Following this dependency rule, we can coarsely reduce video quality by removing NALUs with a DID greater than a specific value. For simplicity's sake, we do not consider Coarse Grain Scalability in the remainder of this paper. However, extending our work to CGS is straightforward.

Temporal Scalability allows to adapt the video frame-rate. The TID specifies the *temporal-layer* of the NALU, i.e., the "frame-rate sub-stream". A NALU belonging to the temporal-layer $tid > 0$ and with $qid = 0$ depends on NALUs of temporal layer $tid-1$, with the same DID and QID parameters. Following this rule, a frame-rate scaling may be accomplished by removing NALUs with a TID greater than a specific value.

Medium Grain Scalability (also known as *progressive refinement*) allows the adaptation of video quality. The video should be encoded with a set of quality enhancement sub-streams, called *quality-layers*. Adding a quality layer reduces the encoding quantization error, and thus improves the PSNR. The QID parameter identifies the quality layer to which a NALU belongs. A NALU belonging to quality layer $qid > 0$ depends on NALUs belonging to quality layer $qid - 1$, having the same DID and TID parameters. Following this dependency rule, quality scaling may be achieved by removing NALUs with a QID greater than a specific value.

Overall, with reference to temporal and medium grain scalability, the dependency rules can be summarized as follows, where the arrows have the meaning of "depends on":

$$\begin{array}{llll} (tid > 0, & qid = 0) & \rightarrow & (tid - 1, \quad qid = 0) \\ (tid \geq 0, & qid > 0) & \rightarrow & (tid, \qquad qid - 1) \end{array}$$

III. The SVEF Evaluation Framework

SVEF is meant to reproduce a distribution chain formed by three actors: streaming server, middlebox and receiver. All actors are connected by an IP network.

Figure 1 shows the structure of SVEF with interactions between single tools and data flows depicted as arrows. The software modules inherited from the JSVM package [11] are represented in grey. The whole process, from the encoding of the original video source to the evaluation after the streaming over a network can be summarized in four steps, better detailed in the following sub-sections:

1) A YUV video is encoded in H.264 SVC format through the JSVM Encoder. The encoded video and its NALU-trace are transferred to the Streamer.
2) The encoded video is transmitted over the IP network by the Streamer, at a fixed frame-rate.
3) In presence of a middlebox, the video NALUs first enter a cross-layer scheduler and then the NALUs are forwarded to the receiver (for example through a wireless link).
4) The Receiver generates in real time a trace of the received NALUs. At the end of the streaming process, the received NALU trace is processed to produce a YUV file (filtered-YUV video) characterized by missing frames due to transmission losses, unsatisfied decoding dependencies or excessive delay. The filtered-YUV video is processed to achieve a simple error concealment,

[1] We remark that the usefulness of the SVEF framework is not limited to the Wireless LAN application that we consider here; as a matter of fact it can be used for several other networked applications. For instance, SVEF can be used to evaluate the performance of overlay multicasting for video applications, where overlay nodes implement cross-layer scheduling

978-1-4244-4671-1/09/$25.00 ©2009 IEEE

obtaining a final-YUV video with the same number of frames as the original video.

A. Video Encoding

We use raw video files stored in the standard YUV format. Video sources are encoded in the H.264 SVC format through the JSVM Encoder. Currently, the framework supports SVC with a single dependency layer and an arbitrary number of quality enhancement layers. From the resulting H.264 encoded video file, we generate an original NALU trace file through the JSVM BitStreamExtractor tool. This trace contains for each NALU the entry shown in Figure 2, where `mem-offset` represents the memory offset from the beginning of the encoded video file up to the current NALU, `NALU-size` represents the length of the current NALU and `Frame-Number` is the number of the video frame to which the NALU belongs. This latter parameter is not provided by JSVM BitStreamExtractor. For this purpose we developed the F-N Stamp module.

B. The Streamer

The Streamer transfers the NALUs over an IP network by parsing the NALU trace and loading data from the H.264 file. For each entry in the NALU trace file, the streamer seeks and loads the corresponding NALU from the H.264 file. This NALU constitutes the payload of a custom layer-5 packet, whose header (see Figure 3) resembles the RTP [13] header, as both have the same size (including payload header). Hence the Streamer feeds the network with packets that have the same length as if a real RTP stack was used. Moreover, with respect to RTP, the custom layer-5 header does not add information that may improve the cross-layer scheduling process, but simply exploits the DID, TID, and QID parameters that are also contained in the RTP payload header. We choose to use a custom header, because we make use of the `mem-offset` information in the generation of the "filtered H.264 video file" described in section III-D.

A layer-5 packet is entirely encapsulated in a UDP packet, which in turn is encapsulated in a set of IP packets. For the Streamer design, we had to choose where to fragment large NALUs to fit into network packets. The two options were fragmentation at the IP layer or at the RTP layer. Fragmentation of IP or RTP packets is similar, with the additional limitation that UDP imposes a maximum payload size of 64 kbytes. Moreover, the NALU SVC header is considered as RTP payload and is thus carried only by the first IP fragment. This means that to perform a cross-layer NALU-based scheduling, a complete reconstruction of the NALU is necessary, both in case of IP or RTP fragmentation.

To limit the programming effort, we choose to fragment at the IP layer, as IP fragmentation/reassembly is a native feature of the Linux kernel. Thus, the Streamer cannot transfer NALUs with a length exceeding 64 kbytes. When some NALUs exceed this threshold, it becomes necessary to re-encode the video enabling the generation of two or more slices per frame.

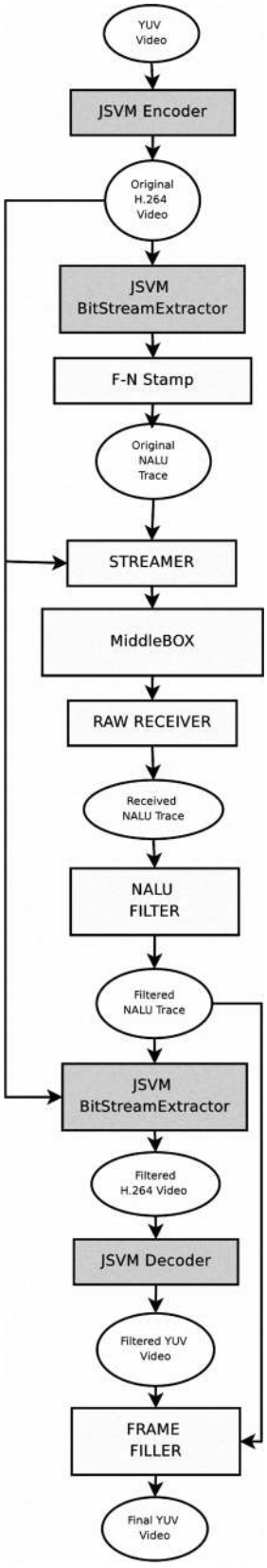

Fig. 1. Software Chain

mem-offset	NALU-size	DID	TID	QID	Frame-Number

Fig. 2. NALU-trace entry

0			31
DID	TID	QID	flags
mem-offset			
NALU-size	Frame-num		

Fig. 3. Layer-5 header

Finally, we observe that the streamer transmits only NALUs of type "SliceData", while the H.264 "ParameterSet" and "StreamHeader" NALUs are provided to the receiver off-line.

C. The Middlebox

The SVEF Middlebox is devised upon a Linux OS to perform cross-layer scheduling. Figure 4 shows the middlebox architecture that we have designed.

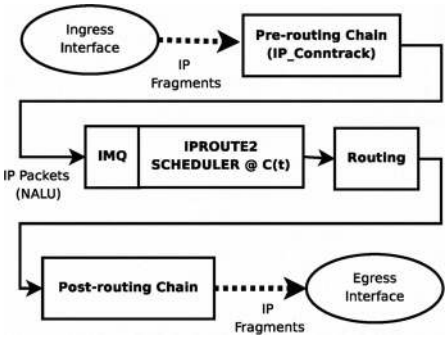

Fig. 4. The Middlebox

When the IP fragments reach the ingress interface, they are reassembled into whole IP packets. For this purpose, the IP_conntrack module of the Linux Kernel is used. When an IP packet (i.e., a NALU) is fully reconstructed, it is transferred to an Intermediate Queue device (IMQ) by means of an Iptables jump. On the IMQ egress we can enforce a custom scheduling policy, taking care that the overall output bandwidth C is equal (or smaller) to the one available on the following network path. Obviously, if this capacity varies over time, the scheduler has to be timely informed by an additional module. The scheduler can operate in a "cross-layer" fashion, since it is able to classify traffic according to the (DID, TID, QID) information contained in the RTP payload header, or in the custom layer-5 header. After exiting the scheduler, the IP packets return to the routing decision module and are fragmented again.

D. Receiver-side Tools

This section describes the tools used at the client side to reproduce a YUV video equivalent to the one that would have been displayed by an H.264 SVC video player. We expect the client to receive the NALUs and dejitter and order them in a play-out buffer, which synchronously provides NALUs to the decoder. The decoder is expected to appropriately discard the NALUs with unsatisfied dependencies (see section II) and also to conceal missing frames. We now describe how our framework reproduces such a chain of client-side operations based on three tools: NALU-Receiver, NALU-Filter and Frame-Filler.

The NALU-Receiver represents the network end-point. It decodes and writes in real time the layer-5 headers of the received packets, thus building a client-side (received) NALU trace file. Moreover, the NALU reception times are recorded through time-stamps.

The NALU trace file is then passed to the NALU-Filter tool that: i) reorders the NALUs according to the sending order, ii) removes NALUs received after the play-out buffer deadline and, iii) removes NALUs with unfulfilled decoding dependencies. The latter cannot be decoded and, moreover, are not handled properly by the current version (v 9.15) of the JSVM Decoder.

The NALU reordering is performed by the NALU Filter on the `mem-offset` field's basis.

To discard NALUs because of excessive delay, the NALU Filter computes a NALU's expected reception time from first NALU's reception time t_0 and frame-rate f as follows: t_0 + f*`frame-number`. When the difference between the reception time and the expected time exceeds a specific play-out delay, the NALU is deleted from the NALU trace file.

After the removal of NALUs with eccessive delay, the NALU Filter discards those NALUs for which the decoding dependencies described in section II are not satisfied (i.e., if a NALU y depends on NALU x and NALU x is not available in the NALU trace-file, then NALU y is deleted).

The resulting (filtered) NALU trace-file is used as a "map" pointing out which NALUs of the original H.264 video file are effectively decoded at the receiving side. We use this map and the original H.264 video file as input to the JSVM BitStreamExtractor tool to obtain a (filtered) H.264 video file. In essence, this is a NALU-subsampled version of the original video file, corresponding to what a hypotetical H.264 SVC client would have decoded and displayed in real time. Then this filtered H.264 video is handed to the JSVM Decoder, which generates a video in uncompressed YUV format.

The filtered YUV video has, in general, fewer frames than the original YUV video because of missing base-layer NALUs. In order to properly compute the PSNR, the Frame-Filler has to conceal the missing frames by copying the previous frame. Missing frames are identified through the `frame-number` field of the filtered NALU trace.

IV. Performance Parameters

Currently, SVEF measures the following performance parameters:

1) number of lost frames,
2) frame-by-frame PSNR, and
3) transmission-efficiency.

The number of lost frames is computed by the Frame-Filler, and the frame-by-frame PSNR is obtained using the JSVM PSNR tool, fed with the original YUV video and the error-concealed YUV video. We report below the definition of

PSNR for frame number n:

$$PSNR(n)_{dB} = 20log_{10}\frac{V_{peak}}{\sqrt{\frac{1}{N_{col}N_{row}}\sum_{i=0}^{N_{col}}\sum_{j=0}^{N_{row}}[Y_S(n,i,j)-Y_D(n,i,j)]^2}}$$

If k is the number of bits per pixel (considering only the luminance component) we have $V_{peak} := 2^k - 1$. The part under the fraction line is the mean square error (MSE) computed from the luminance components Y_S and Y_D of the source image S and of the destination image D.

The *transmission efficiency* (TE) is defined as the ratio of the number of NALUs received by the client which can be usefully decoded (NAL_{useful}) to the total number of received NALUs ($NAL_{received}$).

$$TE = \frac{NAL_{useful}}{NAL_{received}}$$

This measure represents the efficiency of the overall streaming process in exploiting the communication resources. A low transmission efficiency means that most of the received NALUs are useless, and thus their transmission only wasted communication resources. Without a scheduler, NALUs are lost at random and this leads to a considerable number of unsatisfied dependencies at the receiving side and a low transmission efficiency. Conversely, a well-devised cross-layer scheduler should substantially improve the transmission efficiency.

V. Usage of the Framework in a WLAN Scenario

We evaluate the performance of a simple, effective cross-layer scheduler in a WLAN scenario as depicted in Figure 5. All the stations experience optimal channel conditions and the PHY WLAN bitrate is set to 11 Mbps. Under these conditions we measure a maximum UDP bitrate of about 6.3 Mbps. To fully control the packet losses in the middlebox, we throttle its output bandwidth C to 6.0 Mbps (see section III-C). Using this technique, the middlebox becomes the *virtual* bottleneck (VBN) of the communication path and thus no packet loss occurs on the wireless interface (e.g., no overflow of the AP's buffer occurs). The online estimation of C is a fundamental functionality of the middlebox. However, in this paper we are only interested to demonstrate the validity of our SVC streaming framework by means of a cross-layer scheduler in the middlebox. The run-time estimation of C has been dealt with in the literature [15] [16] [17].

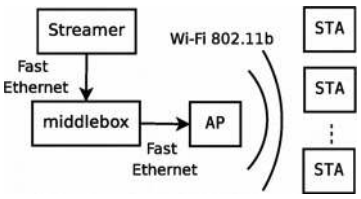

Fig. 5. Experimental testbed

In the middlebox we have devised a cross-layer scheduler with the following goals:

TABLE I
Video test-sequence parameters

BL	MG1	MG2	Full	PSNR
648 kbps	907.3 kbps	1304.7 kbps	2860 kbps	36.64

1) efficient usage of the wireless resource by avoiding the transfer of NALUs that cannot be decoded by the receiver because of unsatisfied dependencies;
2) smooth adaptation of the video quality versus changes in the available capacity C or the offered load of the video traffic.

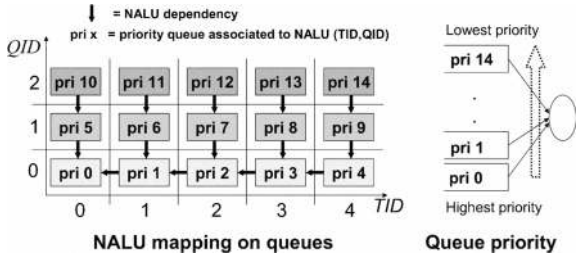

Fig. 6. The Cross-Layer Scheduler

These two goals can be accomplished through a priority queuing discipline (with an overall output rate equal to C), dedicating a *separate* queue to each possible TID-QID combination. Considering that the default range for TID values is from 0 to 4, and considering two additional enhancement quality-layers (i.e., QID values in the range from 0 to 2), $5 \times 3 = 15$ limited-size queues were deployed. Queue #0 has the highest priority and queue #14 has the lowest one, as shown in Figure 6. With such a discipline, an incoming NALU is delivered to queue #n according to the following classification rule:

$$n = 5qid + tid$$

This ensures that an incoming NALU x will have a lower service priority than the NALUs it depends upon (first goal). Moreover, if congestion occurs, the NALUs belonging to higher quality layers will be discarded first, and only afterwards will the NALUs of the base layer be dropped (second goal).

We use a 10 seconds clip of a publicly available 4CIF YUV video (soccer game) at 30 fps. By concatenating 5 repetitions of the video, a 50 second video sequence is generated and then encoded in H.264 SVC format with JSVM, enabling Medium Grain Scalability with three quality-layers (base-layer BL and two enhancement layers MG1 and MG2). The resulting video parameters are summarized in Table I.

A first user starts retrieving the video stream at time 0. Then, a new user arrives every 8 seconds (240 frames) and begins to download the same video stream. Performance parameters are measured on the first user and the play-out delay is fixed to 5 seconds.

Figure 7 shows the Y-PSNR (luminance) over time with and without scheduler. The PSNR is compared with two reference

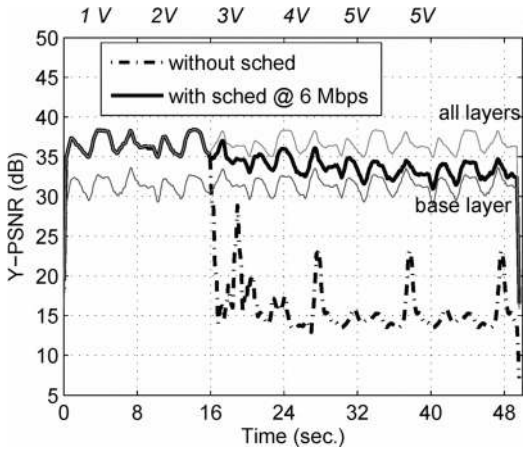

Fig. 7. Y-PSNR of received SVC streams, with and without scheduler, over a mixed network including a WLAN at 11 Mbps

TABLE II
SUMMARY PERFORMANCE

Scenario	TX Efficiency	# Missing Frames	Average PSNR
with sched	100.00 %	0/1490	34.67
no sched	54.64 %	941/1490	22.52

curves: i) the ideal PSNR (top curve labeled "all layers") of the stream for the case of no NALU loss, where the resulting PSNR depends only on the degradation due to the encoding process, and ii) the PSNR provided by the base layer only (labeled "base layer"), assuming that all base layer NALUs are received and all NALUs of other layers are dropped. Figure 7 confirms that *the delivery performance of H.264 SVC is poor without cross-layer scheduling*, i.e., when MAC frames, and as a consequence NALUs, are dropped randomly due to the overloading of the AP buffer. The resulting video frequently "freezes" due to frames lost, and therefore, the overall video quality is unacceptable. With an average total video rate of 2.86 Mbps, this happens when three streams are delivered. The PSNR does not degrade further when additional streams are admitted. This is due to the fact that the PSNR given by the comparison of two random frames from the same test sequence is around 15 dB, as confirmed by further experiments (not shown here). Thus, this is the lowest PSNR value that we can expect.

In Table II, we provide some summarizing results on the delivery efficiency for the previously described experiments. The most interesting result is that the transmission efficiency of the considered scheduler is 100%. Without the scheduler, transfer efficiency is very poor. The column reporting the number of missing video frames gives an estimate of the perceived quality of the final video stream (refer to the web site [14] for a visual comparison of the actual streams).

VI. CONCLUSION

H.264 scalable video coding is a highly interesting for video streaming services, especially when the bandwidth available to a stream is variable over time. For instance, when users share a wireless medium, the available bandwidth changes with the number of users. An evaluation framework is a fundamental instrument to fine tune the encoding parameters and the scheduling technique in a specific service environment. We provide it as a set of open-source tools and assess its effectiveness in evaluating the performance of a cross-layer scheduler in a WLAN hot-spot scenario.

ACKNOWLEDGMENT

The author Andrea Detti has been founded by the Italian research program PRIN 2007 within the "SORPASSO" project.

REFERENCES

[1] IEEE Standard 802.11-2007, IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications; June 2007.

[2] K. Ramachandran, H. Kremo, M. Gruteser, P. Spasojevic, I. Seskar, "Scalability Analysis of Rate Adaptation Techniques in Congested IEEE 802.11 Networks: An ORBIT Testbed Comparative Study", IEEE WoWMoM 2007.

[3] Chuan Heng Foh, Yu Zhang, Zefeng Ni, Jianfei Cai, King Ngi Ngan, "Optimized Cross-Layer Design for Scalable Video Transmission Over the IEEE 802.11e Networks", IEEE Transactions on Circuits and Systems for Video Technology, Volume 17, Issue 12, Dec. 2007

[4] M. van der Schaar, S. Krishnamachari, Choi Sunghyun, Xu Xiaofeng, "Adaptive cross-layer protection strategies for robust scalable video transmission over 802.11 WLANs", IEEE Journal on Selected Areas in Communications, Volume 21, Issue 10, Dec. 2003, pp. 1752-1763

[5] ITU-T recommendation H.264: Advanced video coding for generic audiovisual services, International Telecommunications Union, November 2007.

[6] R. Kuschnig, I. Kofler, M. Ransburg, H. Hellwagner, "Design options and comparison of in-network H.264/SVC adaptation", Journal of Visual Communication and Image Representation, Volume 19, Issue 8, December 2008, pp. 529-542.

[7] H. Liqiao, D. Raychaudhuri, Liu Hang, K. Ramaswamy, "Cross layer optimization for scalable video multicast over 802.11 WLANs", 3rd IEEE Consumer Communications and Networking Conference, Jan. 2006

[8] Y. P. Fallah, P. Nasiopoulos, H. Alnuweiri, "Efficient Transmission of H.264 Video over Multirate IEEE 802.11e WLANs", EURASIP Journal on Wireless Communications and Networking, Volume 2008

[9] I. Kofler,M. Prangl, R. Kuschnig, H. Hellwagner, "An H.264/SVC-based adaptation proxy on a WiFi router", Proceedings of the 18th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), Braunschweig, Germany, May 2008, pp. 63-68.

[10] M. Eberhard, L. Celetto, C. Timmerer, E. Quacchio, H. Hellwagner, F.S. Rovati, "An interoperable streaming framework for Scalable Video Coding based on MPEG-21", 5th International Conference on Visual Information Engineering, Aug. 2008. VIE 2008, pp.723-728

[11] Joint Scalable Video Model - reference software: http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm

[12] Th. Zahariadis, "ASTRALS Project presentation", IBC 2007, Amsterdam, September 2007

[13] S. Wenger, Y.-K. Wang, T. Schierl, and A. Eleftheriadis, "RTP payload format for SVC video", Internet Draft, draft-ietf-avt-rtp-svc-17, February 2009, work in progress.

[14] SVEF - reference software: http://netgroup.uniroma2.it/Andrea_Detti/SVCEvalFrame/

[15] H. K. Lee, V. Hall, K. H. Yum, K. Kim, E. Kim, "Bandwidth estimation in wireless lans for multimedia streaming services", IEEE International Conference on Multimedia and Expo (ICME), 2006.

[16] M. Neilsen, K. Ovsthus, and L. Landmark, Field trials of two 802.11 residual bandwidth estimation methods, IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), 2006.

[17] C. Sarr, C. Chaudet, G. Chelius, and I. Lassous, "A node-based available bandwidth evaluation in ieee 802.11 ad hoc networks", 11th International Conference on Parallel and Distributed Systems (ICPADS), 2005.