NextSharePC: An Open-Source BitTorrent-based P2P Client Supporting SVC

Michael Eberhard
Alpen-Adria-Universitaet Klagenfurt
Universitaetsstrasse 65-67
Klagenfurt, Austria
michael.eberhard@aau.at

Amit Kumar
STMicroelectronics
Plot no. 1, Knowledge Park III
Greater Noida, India 201308
amit.kumar-ast@st.com

Licio Mapelli
STMicroelectronics
Agrate Brianza, Via Olivetti
Milan, Italy
licio.mapelli@st.com

Andi Palo
University of Rome Sapienza
Department of Computer and System Science
Via Ariosto 25, Rome, 00185, Italy
andi@dis.uniroma1.it

Riccardo Petrocco
Technische Universiteit Delft
Mekelweg 4
Delft, The Netherlands
r.petrocco@gmail.com

Mikko Uitto
VTT Technical Research Centre of Finland
Kaitoväylä 1
90571 Oulu, Finland
mikko.uitto@vtt.fi

ABSTRACT

In this paper the open-source NextSharePC demo is presented, which allows to distribute layered video content over P2P networks. The cost-efficient distribution of multimedia content over P2P networks has become very popular in recent years. Furthermore, multimedia content is consumed on a variety of devices, which requires to provide content in different qualities. By distributing layered content over P2P networks the NextSharePC demo provides both, cost-efficient distribution and a video quality suitable for various devices. The different modules of NextSharePC and their usage are described in detail in this paper. To the authors' knowledge the NextSharePC demo is the first open-source P2P implementation with full SVC support.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architectures

Keywords

P2P, SVC, open source, multimedia, streaming

1. INTRODUCTION

In this paper an open-source P2P system with full support for the *Scalable Video Coding* (SVC) [12] of the *Advanced Video Coding* (AVC) standard is presented. The utilized P2P system is the *NextShare* system, which is based on the *BitTorrent* protocol and has been developed within the *P2P-Next* project [3]. The usage of P2P systems for streaming multimedia content provides a popular alternative to multimedia portals, as it allows the distribution of multimedia content to thousands of users without requiring an expensive distribution infrastructure. The usage of SVC ensures that the user can easily select the desired streaming quality and that changes in network conditions can be handled during the streaming sessions.

SVC provides scalable video streams, which are composed of a base layer and one or more enhancement layers, where each enhancement layer can improve the temporal rate, the spatial rate, or the quality of the video content. The scalable properties of SVC refer to the capability of adapting the bitstream to varying terminal capabilities, network conditions and the end user preferences by selectively discarding parts of the scalable bitstream and still obtaining a bitstream which can be decoded.

When using P2P, the usage of SVC is clearly better suited than distributing the same content in multiple qualities, as all peers can share at least the lower layers with each other, while only peers streaming exactly the same quality can share content when using multiple files to provide different qualities.

This paper describes the demo prototype called NextSharePC, which is provided as open-source [3]. In Section 2, the producer modules used for encoding and ingesting layered content into the P2P system are described. In Section 3, the consumer modules used for streaming and consuming the layered content from the P2P system are presented. Section 4 provides an overview of the usage of the demo prototype, while Section 5 describes the demo setup. Finally, Section 6 concludes the paper.

2. PRODUCER MODULES

The producer modules are used to encode the content and prepare it for ingest into the P2P system.

2.1 Encoder

The encoded bitstream contains a series of data packets called *Network Abstraction Layer Units* (NALUs), each con-

MMSys '12, February 22-24, 2012, Chapel Hill, North Carolina, USA.

taining either actual video data, or data required for proper decoding of the bitstream, like *Sequence Parameter Sets* (SPS) and *Picture Parameter Sets* (PPS) or *Supplemental Enhancement Information* (SEI). All NALUs belonging to one image for a time instant form an *Access Unit* (AU). In order to map NALUs to input pieces for the P2P engine, a rate control is required to adapt the intrinsic variability of the bitstream rate for the packetization process. Thus, the JSVM SVC encoder [5] has been extended with a *Constant Bit-Rate* (CBR) algorithm. The CBR algorithm uses the *Quantization Parameter* (QP) value in order to adapt the variable rate bitstream for a limited bandwidth channel. The CBR algorithm applied for the presented system uses a buffer-based CBR control method proposed in [14], which is based on buffer management and is suitable for multiple layer coding. The CBR algorithm tries to achieve at the end of each Intra period the same buffer fullness that was available before encoding the last Intra picture. As a consequence, it performs a constant bitrate encoding since every Intra period of IDR length consists of roughly the same number of bits:

$$\text{Target IDR Size} = \text{AvgBitPict} \times \text{IDR Length} \quad (1)$$

where *AvgBitPict* is the average amount of bits per picture obtained as the ratio of the target bitrate and the sequence's frame rate. In order to produce a nearly constant amount of bits for the *Target IDR size*, the algorithm checks the buffer fullness and compares the current buffer occupancy level with the target level. In case the buffer is too empty, the QP is increased, while in case of a possible overflow, the QP is decreased and the filler NALUs are inserted. The encoder along with the buffer-based CBR is able to achieve good bit-rate control performance and to maintain uniform image quality throughout the sequence at the same time. As the CBR still produces small bitrate fluctuations [14], these fluctuations are considered while creating the base and enhancement layer pieces as described in the following section.

2.2 Ingest

The ingest modules take the encoded audio and video content as input and prepare the content for P2P distribution.

2.2.1 NALU Splitter

The functionality of the *Splitter* is similar to a demuxer. The basic idea behind the Splitter application is to create a separate file for each SVC layer. This allows to distribute the files separately in NextShare, which is important for easy selection of the desired target quality. 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 BitTorrent pieces (or GOPs) belonging to the same layer. The input parameters for the Splitter are the number of frames per piece and the piece size. When a NALU does not fit into the block it is dropped. If a piece still contains empty space after all NALUs have been fitted in, a filler NALU is appended in order to maintain the constant block size.

As the CBR algorithm of the SVC encoder produces small bitrate fluctuations, the Splitter's input parameters are tuned to avoid NALU dropping while creating the file. Thus, the target bitrate during encoding is chosen a bit lower than the bitrate which would fit into the desired piece size. If the CBR algorithm produces a higher bitrate, this ensures that all desired NALUs can still be fit into the piece. In case of a lower bitrate, a filler NALU is used for padding. Although this results in a slight overhead in piece size (∼2% of the piece are used for padding), the overhead ensures that no NALUs are dropped during ingest.

2.2.2 Muxer

After splitting the SVC elementary stream into base and enhancement layers, the base layer and the audio track are forwarded to the *Muxer* module. The video and audio content are split in pieces, each piece belonging to a specific time slot (in case of SVC, each piece contains a closed GOP). Each GOP of the video base layer is multiplexed with it's audio part for every time slot. As the base layer file and the audio track have a costant bitrate, the piece for each time slot will contain the same number of bits. The only variable factor will be the muxing overhead and the consecutive padding. Each time slot typically contains audio and video content with a length of 2.56 seconds (64 frames at 25 fps).

To provide the audio and video content together in a suitable container format, there are many different transport stream containers available. The NextShare architecture does not require a specific container format, but for the system's implementation the MPEG-4 File Format (MP4) is used as container format. MP4 is a popular container format and is supported by almost all media players and devices. Furthermore, the intrinsic overhead that MP4 introduces is low compared to the MPEG-2 Transport Stream (MPEG-TS), e.g., in the presented system the overhead due to the multiplexing is about 7% (and would be ∼50% for MPEG-TS). To consider the MP4 container's overhead, the bitrate during encoding is again selected lower than possible for the selected piece size. If all of the allocated overhead is not required for muxing, the remaining space of the piece is padded with MP4 empty boxes in order to reach a constant piece size. This results in a self-contained backwards-compatible media file for every time slot, which can be processed also by media players only supporting AVC. In the demo prototype MP4-Box [1], which is available as open-source, was used for multiplexing.

2.2.3 Metadata Creator

The *Metadata Creator* extracts the properties of the SVC layers and the parameter sets from the SVC bitstream and provides them as a Session Description Protocol (SDP) file to the media player. To gather the relevant attributes for the SVC layers, the metadata creator analyzes the Scalability Information SEI Message (SSEI) at the beginning of the SVC bitstream. The SSEI contains for every layer a number of attributes, including the bitrate, the framerate, and the spatial resolution, which are extracted by the Metadata Creator. Furthermore, the Metadata Creator extracts the SPS and PPS elements. To provide this information to the media player, the extracted information is wrapped into an SDP file formatted according to [10], which allows to signal the properties of the scalable layers and their dependencies on each other. The SDP file is subsequently provided to the media player before the streaming process starts, to ensure that the *De-Packetizer* and *Decoder* have all the required information to initialize their modules.

2.2.4 *Content Distribution*

Once the files containing the different layers have been created, they are saved in a specific directory along with the metadata file. Subsequently, a torrent file for that specific directory is created, which is fully compliant with the Bittorrent specification [6], and can be distributed by every BitTorrent compliant tracker. Utilizing the torrent file, the content is split into pieces of equal size and can be distributed in the NextShare P2P system. As each layer is provided in a separate file, the users can decide their target playback quality by selecting which files to download.

3. CONSUMER MODULES

This section describe how the layered content is downloaded from the P2P system and how the processing chain until the presentation of the video works.

3.1 Module initialization

In the current implementation of the system, modules are initialized following an iterative process.

Figure 1 shows the initialization sequence of the three independent modules, consisting of the *P2P Client*, the *Request Interface*, and the *Media Player*. The user interacts with the Media Player, for which a modified version of the VLC Media Player [4] is utilized. Once the user presses the play button, the P2P Client is initialized, or alternatively, if the engine is already running and listening to incoming requests, a *start* command followed by the torrent location triggers the content retrieval. The P2P engine assigns a unique RTP address to the Media Player, which is used to set the player's listening channel for incoming data. Assigning a unique address for each Media Player instance allows the P2P engine to participate in multiple swarms, allowing the simultaneous playback of different videos.

The Request Interface module is than initialized with the following parameters:

- A local HTTP address from which the Request Interface will be receiving the downloaded layers from the P2P engine.
- The RTP address previously provided to the Media Player, which is utilized by the Request Interface module to provide the final data stream to the player.

After the modules have been initialized, The P2P Client initializes a local HTTP server that will respond to the incoming requests from the Request Interface module. The P2P Client provides data to the Request Interface only after an initial buffer has been downloaded, usually set to 20 seconds of stream containing at least the base layer pieces. Once the initial buffer has been filled, the P2P engine starts replying to the HTTP requests of the Request Interface, providing a piece with content for one time slot in response to every request. This behavior provides a certain level of synchronization between the involved modules, considering that once the Request Interface's and Media Player's buffers are filled, data will be requested at constant intervals, due to the constant bitrate encoding.

The presented initialization phase can also be performed by launching the modules independently. For using the demo in test environments, an instance of the P2P engine acting as seeder and content provider is firstly initialized. Afterwards, the consumer modules are initialized with the

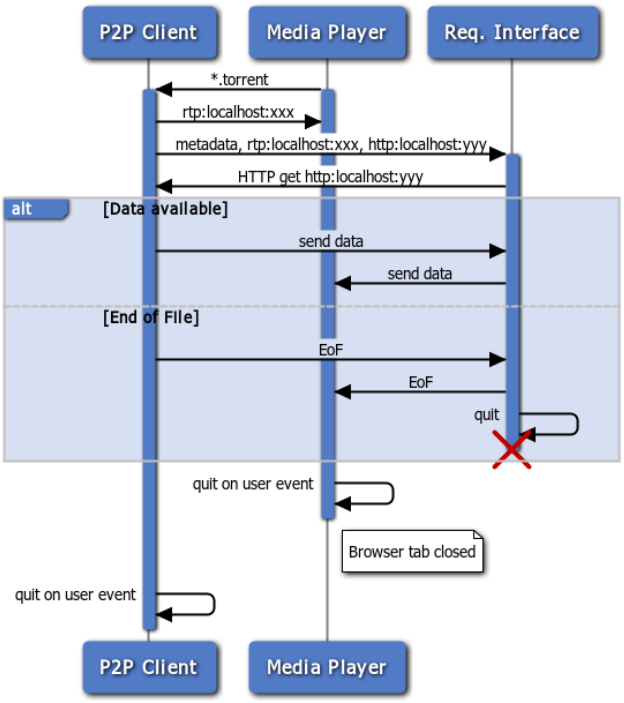

Figure 1: Sequence Diagram for Consumer Modules

required parameters, preserving the order previously discussed and presented in Figure 1.

In the following, each of the modules on the consumer side are described in more detail.

3.2 P2P Client

The P2P Client is the module responsible for downloading the pieces of the layers and providing them in correct order to the Request Interface module. Upon receiving the torrent file, the P2P Client detects the order of the different layers, which are distributed in separate files, and schedules the algorithm responsible of retrieving the data, which is called *Piece Picker*.

The Piece Picker starts downloading the stream as soon as an initial set of peers has been contacted, following the algorithm presented in [7]. Once data has been retrieved and verified using the cryptographic hashes provided in the torrent file, the P2P engine stores the data on disk, allowing the retransmission to other peers and offline playback capabilities, and fills a data stream used by the HTTP server to send data to the Request Interface module.

During the development of the system, several piece-picking algorithms, which try to provide the best possible playback quality for the given network conditions, have been investigated and integrated into the P2P engine. Details for the piece-picking algorithms are provided in [8] and [9].

3.3 Request Interface

As illustrated in Figure 1, the Request Interface module requests pieces from the P2P Client and forwards them to the media player. In this section the two major parts of the module, the *Demuxer* and the *RTP Packetizer*, are described in detail.

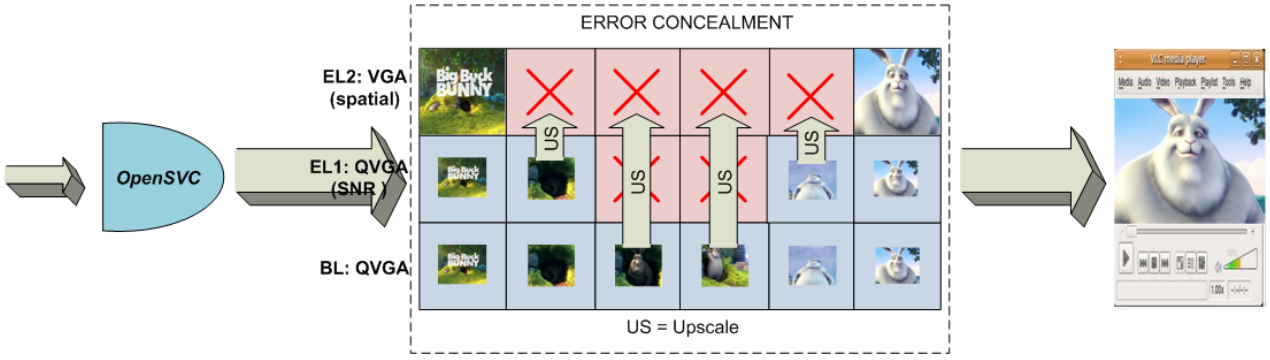

Figure 2: The Decoding Process using Error Concealment

3.3.1 *Demuxer*

The demuxer is only used for the processing of base layer pieces, which have been previously wrapped into the MP4 file format to ensure backwards-compatibility to a variety of media players. After receiving a base layer piece, the demuxing of the MP4 file format is performed. The minimum number of pieces retrieved from the background process for each time slot is one, corresponding to the multiplexed base layer track together with audio track. If also enhancement layer pieces are received, these are directly forwarded to the RTP Packetizer module. The Demuxer extracts the audio and video content by demultiplexing the piece. Again, MP4-Box was utilized for this task. The extracted video part contains 2.56 seconds of AVC video content while the extracted audio part contains 2.56 seconds of MP3 audio content. Subsequently, the Demuxer module passes the extracted content to the RTP Packetizer module.

3.3.2 *RTP Packetizer*

This module is composed of two sub-modules: one for streaming the video content and one for streaming the audio content to the Media Player. The audio and video streams use two different RTP destination ports and thus represent two separate sessions.

The video streaming sub-module is responsible for creating RTP packets from an elementary SVC stream. This module is compliant with the RTP protocol [11], and it supports Single Session Transmission Mode with Non-Interleaved Packetization Mode (STAP-A, FU-A), as described in [16]. The module streams the base layer and and all available enhancement layers to the Media Player. It is invoked as a synchronous function call from the Demuxer module after the latter has demuxed the video content. The timestamps for every RTP packet are increased monotonically.

The audio streaming sub-module is a standalone process. It utilizes the open source Live555 library, which provides implementations for multiple streaming protocols. The process listens for incoming data on its pipe. As soon as the data available on the pipe is enough to construct an RTP packet, it retrieves the content from the pipe and sends a new packet to the Media Player. The RTP packets are sent in a burst when sufficient data is available on the pipe and does not transmit anything for the remainder of the time lot. This bursts are compensated for playback by the Media Player's receiver buffer.

3.4 Media Player

In this section the media player is described in detail. As basis for the implementation, the VLC Media Player is utilized.

3.4.1 *De-Packetizer*

As the Packetizer module is creating two different RTP sessions for audio and video, the *De-Packetizer* is listening at two different RTP port in a loop with a buffering time set at initialization. The buffering time allows the De-Packetizer to buffer a certain amount of content before the stream is sent to the Decoder. The RTP De-Packetizer is using the modified live555 library to support SVC files, which is integrated into the VLC. The VLC is initialized by using the SDP file from the Metadata Creator as input. It retrieves the various input parameters to de-packetize and decode the audio and video stream. The live555 library is modified to initialize the decoder with the parameter sets (SPS and PPS) of the highest layer. If not all layers of the SVC file are received, the *Error Concealment* module integrated with the *Decoder* is utilized to upscale the video to the desired resolution.

3.4.2 *Decoder*

To decode the SVC stream after de-packetization, the *OpenSVC* decoder [2] has been integrated into VLC. OpenSVC is integrated as decoding library into VLC media player to be used for decoding and rendering of SVC content. The OpenSVC decoder has been chosen as it provides real-time decoding capabilities even on less powerful machines. For the demo prototype, the OpenSVC and VLC media player have been tested for SVC streams with up to 3 enhancement layers and SD playback quality, which works in real-time on casual laptops.

3.4.3 *Error Concealment*

The error concealment strategies for the SVC Decoder are critical in error-prone video transmission conditions. The target for the concealment is not only to prevent the Decoder from crashing, but also to provide sufficient quality of experience (QoE) for the end user. Currently, many of the implemented SVC error concealment algorithms are designed from the basics of the AVC decoder, including, e.g., pixel-value interpolation and frame copying of correctly received video data for the concealment [13] [15].

The error concealment scenarios in the presented P2P SVC chain differs from the usual setup regarding the man-

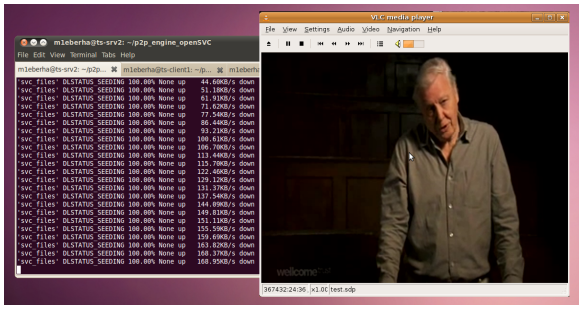

Figure 3: Demo Screenshot

ner how packet losses occur in traditional end-to-end video streaming. One piece containing a GOP is always received in the current P2P chain, meaning that random packet/frame losses are not possible. Thus, if pieces are lost the Decoder receives either a GOP with a spatially lower resolution (if spatial scalability is used) or a GOP with lower quality (if quality scalability is used).

The SVC standard defines that the decoder is able to handle the spatial or quality degradation in the middle of the sequence as long as the GOP structure follows the standard. Usually this means that IDR pictures are used to break the decoding chain between the GOPs (i.e., only closed GOPs are produced during encoding). On the player side, such as in VLC, this can be a challenge when using spatially scalable videos, since the resolution can vary during the playback. To prevent fluctuations in the spatial resolution, a picture upscaling functionality was implementation as an error concealment method and as a separate block between the Decoder and VLC player. By using the upscaling, the Media Player always receives the video with the target resolution during playback.

The implemented error concealment method monitors the resolution of the first IDR picture in the received content for the time slot. If it does not receive sufficient enhancement layers to reach the target resolution, the upscaling function is activated and the low-resolution picture (usually the base layer) is upscaled into the target resolution by utilizing 4-tap integer-based filters. The upscaling causes some blurriness to the high-resolution picture which can be reduced by using a higher bitrate for the base layer.

The decoding process while using error concealment is illustrated in Figure 2. After the NAL unit is decoded with the OpenSVC Decoder, the error concealment block checks whether the frame is ready to be forwarded to the Media Player. If this is the case, but the received resolution does not match the target resolution, the frame is forwarded to the Error Concealment module.

4. USAGE

The provided demo prototype consists of two parts: the producer tools allow to encode and packetize the content and subsequently seed the content in the P2P system. The consumer tools are used to download and play the content from the P2P network.

At the producer side, the raw video files firstly need to be encoded using the SVC Encoder. The SVC Encoder takes the configuration file, which allows to specify the layer properties, as input. Subsequently, the SVC Splitter is used to create separate files for each layer. The SVC Splitter takes the desired pieces size in bytes and the frames per piece as input parameter. The Metadata Creator parses the header of the SVC stream to extract the relevant data and create the SDP file. After the files are created, the base layer is muxed with the audio content using MP4-Box. Finally, the torrent file is created using the script *create_svc_torrent* and the P2P Engine is started to seed the content to other peers.

On the consumer side, the P2P Client, the Request Interface, and the Media Player are all started. The P2P Client utilizes the torrent file to connect to the seeding P2P Engine. The Request Interface is started using the ports for communicating with the P2P Client and the Media Player as parameters. Finally, the Media Player is started using the port for communicating with the Request Interface and the SDP file as parameters. Once the user decides to start the playback, the Media Player initializes the communication chain as illustrated in Figure 1. In addition, the VLC Media Player uses the following parameters:

- --reset-plugins-cache: resets the cache
- --reset-config: resets the vlc configuration
- -demux h264: enables SVC decoding
- --h264-svc-force-video: forces all the possible late frames to be displayed
- --h264-svc-error-concealment <VALUE>: enables/disables error concealment
- --buffering-time <VALUE>: buffering time for the Live555 library

When the user triggers the play command of the VLC Media Player, the playback is started soon afterwards.

5. DEMO SETUP

In the demo setup, a part of the producer tools and the entire consumer chain are shown. A screenshot of the running demo is provided in Figure 3. At the producer site, the SVC content has already been prepared for ingest, as real-time SVC encoding in high quality is not possible with the optimized JSVM encoder. Thus, the seeding of the content into the NextShare P2P system is demonstrated. The console displayed in Figure 3 shows the download bitrate provided by the seeder to the P2P client consuming the content. At the consumer site, the P2P client, which downloads the pieces for the different layers based on the network conditions, is shown. The P2P client downloads the pieces prioritized by their playback deadlines and layers [8][9]. Changes to the network conditions can optionally be emulated using a bandwidth simulator, to ensure that dynamic network conditions can be simulated even if the demo is shown on a single machine. The pieces downloaded by the P2P client are requested by the Request Interface in time for playback and forwarded to the Media Player. Furthermore, the Media Player is started, which performs the playback of the content and changes the playback quality dynamically based on the pieces provided by the Request Interface. The playback of the media player is also shown in Figure 3.

6. CONCLUSION

In this paper a demo prototype for distributing and consuming SVC content over a BitTorrent-based P2P system was presented. In Section 2, the producer modules are presented. The SVC Encoder encodes the raw video content to SVC utilizing a CBR algorithm. For ingest, the encoded bitstream is split into files for each layer, the base layer is muxed with the audio content, and the metadata required for playback is extracted. Finally, the content is split into pieces and seeded by the P2P Engine for download by other peers.

In Section 3 the consumer tools are presented. The P2P Client downloads the pieces from the seeding peer while prioritizing lower layer pieces. The Request Interface requests the pieces from the P2P Client, demuxes the base layer, and provides the audio and video content to the Media Player as RTP streams. The Media Player contains the RTP De-Packetizer, the SVC Decoder, and the Error Concealment module. The Media Player triggers the playback, de-packetizes the RTP packets, forwards the video content to the SVC Decoder, and uses the Error Concealment module to upscale the video content in case the spatial enhancement layers are not received in time for playback.

The implementation of the SVC/P2P prototype was performed to show how the quality of streaming video content over P2P can be improved by using SVC. The prototype is to the authors' knowledge the first open-source P2P prototype with full SVC support.

7. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant agreement no 216217 (P2P-Next).

8. REFERENCES

[1] GPAC. URL: http://gpac.wp.institut-telecom.fr/. Last accessed 12-December-2011.

[2] Open SVC decoder. URL: http://sourceforge.net/projects/opensvcdecoder/. Last accessed 12-December-2011.

[3] The P2P-Next project. URL: http://www.p2p-next.org. Last accessed 12-December-2011.

[4] VLC media player. URL: http://www.videolan.org/vlc/. Last accessed 12-December-2011.

[5] *JSVM 9.15 Software Manual*, 2009.

[6] B. Cohen. BitTorrent Protocol 1.0. URL: http://www.bittorrent.org. Last accessed 12-December-2011.

[7] M. Eberhard, A. Kumar, S. Mignanti, R. Petrocco, and M. Uitto. A framework for distributing scalable content over peer-to-peer networks. *International Journal On Advances in Internet Technology*, 4(1&2):1–13, 2011.

[8] M. Eberhard, T. Szkaliczki, H. Hellwagner, L. Szobonya, and C. Timmerer. Knapsack problem-based piece-picking algorithms for layered content in peer-to-peer networks. In *Proceedings of the 2010 ACM workshop on Advanced Video Streaming Techniques for Peer-to-Peer Networks and Social Networking*, AVSTP2P '10, pages 71–76, New York, NY, USA, 2010. ACM.

[9] R. Petrocco, M. Eberhard, J. Pouwelse, and D. Epema. Deftpack: A robust piece-picking algorithm for scalable video coding in P2P systems. In *Proceedings of the International Symposium on Multimedia 2011*, ISM'11, to be published.

[10] T. Schierl and S. Wenger. Signaling media decoding dependency in the session description protocol. RCF 5538, 2009.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 3550: A Transport Protocol for Real-Time Applications (RTP), 2003.

[12] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. 17(9):1103–1120, Sept. 2007.

[13] S.Kumar, L.Xu, M.K.Mandal, and S.Panchanathan. Error resiliency schemes in H.264/AVC standard. *Elsevier Journal of Visual Communication and Image Representation*, 17(2):425–450, 2006.

[14] T.Anselmo and D.Alfonso. Buffer-based constant bit-rate control for scalable video coding. PCS 2007, 2007.

[15] M. Uitto and J.Vehkaperä. Spatial enhancement layer utilisation for SVC in base layer error concealment. In *Mobimedia '09 Proceedings of the 5th International ICST Mobile Multimedia Communications Conference*, London, United Kingdom, 2009.

[16] S. Wenger, Y.-K. Wang, T. Schierl, and A. Eleftheriadis. RTP Payload Format for Scalable Video Coding, 2011.