A Framework for Distributing Scalable Content over Peer-to-Peer Networks

Michael Eberhard*, Amit Kumar†, Silvano Mignanti‡, Riccardo Petrocco§, Mikko Uitto¶
*Klagenfurt University, Klagenfurt, Austria, michael.eberhard@itec.uni-klu.ac.at
†STMicroelectronics, Milan, Italy, amit-agr.kumar@st.com
‡University of Rome Sapienza, Rome, Italy, silvano.mignanti@dis.uniroma1.it
§Technische Universiteit Delft, Delft, The Netherlands, r.petrocco@gmail.com
¶VTT Technical Research Centre, Oulu, Finland, mikko.uitto@vtt.fi

***Abstract*—Peer-to-Peer systems are nowadays a very popular solution for multimedia distribution, as they provide significant cost benefits compared with traditional server-client distribution. Additionally, the distribution of scalable content enables the consumption of the content in a quality suited for the available bandwidth and the capabilities of the end-user devices. Thus, the distribution of scalable content over Peer-to-Peer network is a very actual research topic. This paper presents a framework for the distribution of scalable content in a fully distributed Peer-to-Peer network. The architectural description includes how the scalable layers of the content are mapped to the pieces distributed in the Peer-to-Peer system and detailed descriptions of the producer- and consumer-site architecture of the system. Additionally, an evaluation of the system's performance in different scenarios is provided. The test series in the evaluation section assess the performance of our layered piece-picking core and provide a comparison of the performance of our system's multi layer and single layer implementations. The presented system is to our knowledge the first open-source Peer-to-Peer network with full Scalable Video Coding support.**

***Keywords*-Peer-to-Peer; Scalable Video Coding; Packetizing; Error Concealment; Performance Evaluation**

I. INTRODUCTION

The streaming of content over Peer-to-Peer (P2P) networks becomes more important as the popularity of Internet multimedia services is increasing and the corresponding server costs are rising. One of the major challenges of distributing multimedia content is that different users often require the content in different quality. On the one hand, this is due to the differences in the user's network connections, which can differ depending on the user's location during the content consumption. On the other hand, the users consume the content on various terminals like TV sets or mobile devices, which have different capabilities in terms of resolution, processing power, or power supply.

These problems are addressed by layered streaming systems that provide the content in different qualities within a single bitstream. The architecture of our P2P streaming system supporting scalable content has been originally published in [7]. This paper extends the previous system description and provides an additional evaluation of the system's performance. In this paper we are going to describe our entire framework for the distribution of scalable content in a fully distributed P2P network. The P2P system targeted for the integration is the NextShare system, which is developed within the P2P-Next project [3]. P2P-Next is a research project partially founded by the European Commission in the context of the Framework Program 7, within the ICT (Information and Communication Technology) theme. The main goal of P2P-Next is the development of an open-source next generation P2P content delivery platform, the NextShare system.

The NextShare system has been developed based on the Bittorrent protocol [1] and thus provides an implementation of a fully distributed P2P system. To support Video on Demand (VoD), live streaming and the distribution of scalable content in the NextShare system, a number of modifications to the original Bittorrent protocol have been performed [11], as the original Bittorrent protocol does not support streaming. The scalable codecs used within NextShare are based on the Scalable Video Coding (SVC) extension of the Advanced Video Coding (AVC) standard [14].

One of the main reasons for implementing SVC support for the NextShare system is that there is to our knowledge today no open-source P2P system supporting SVC available that can be downloaded and tested by interested users. The advantages of distributing scalable content compared to simulcast approaches have been evaluated in a number of surveys (see, e.g., [15]). Additionally, we provide a comparison to our implementation for single layer content in Section VI-B to illustrate the advantages of using scalable content.

The remainder of this paper is organized as follows: Section II provides an overview of the related work. In Section III, the approach for the integration of the scalable content into the NextShare system is described. In the following two sections, the producer- and consumer-site of this architecture are described in detail. Section VI provides an evaluation of our implementation in terms of piece download efficiency as well as a comparison to the traditional single layer approach. Finally, future work is addressed in Section VII and Section VIII concludes the paper.

II. Related Work

The distribution of multimedia content over P2P networks has been a popular research topic in recent years. Due to the increasing popularity of streaming high-quality multimedia content over the Internet, P2P provides a cost-efficient alternative to reduce server costs.

The distribution of layered content over P2P systems has also been addressed in the literature before. LayerP2P [10] provides a well defined solution for distribution SVC content over P2P, but does not utilize real SVC codecs for the prototype implementation and relies on the usage of H.264/AVC-compatible codecs that can only be used to test one of SVC's scalability dimensions, the temporal scalability. Thus, one of the goals of the NextShare implementation was to design, implement, and distribute an open-source system with full SVC support. Other systems supporting the distribution of SVC content over P2P are described in [12] and [8]. PALS [12] provides a receiver-driven solution for receiving layered content over P2P. [8] describes how SVC can be integrated into a tree-based P2P system. However, both approaches to not allow an easy integration into existing P2P systems, as the implementations have been based on proprietary systems and protocols. Regarding compatibility an advantage of our implementation is that it has been based on the wide spread Bittorrent protocol and all architectural choices have been performed while ensuring backwards compatibility to existing Bittorrent clients. This allows an easy integration of the new scalable video technology into existing P2P communities. Furthermore, backwards-compatibility of the base layer for existing Bittorrent clients is provided.

III. NextShare Integration

To fully integrate scalable content into the NextShare system, a number of problems had to be addressed. Two main problems, the selection of suitable scalability layers and the mapping of the layers to Bittorrent pieces, are described in detail within this section. While the selection of the scalability layers tries to consider all popular qualities and to support a number of different network connections, the mapping to the scalability layers to the Bittorrent pieces tries to ensure that the best trade-off between flexibility for possible quality switches and overhead in terms of piece management is found.

It should be noted that even though we are using SVC within our NextShare system, all design decisions have been made with the intention to make the architecture codec-agnostic. Thus, if another scalable video codec is utilized within the NextShare system, only the coding and packaging tools need to be replaced, while the integration into the NextShare core will remain suitable for every other layered codec.

Table I
Scalability Layers

Bit Rate	Resolution	Quality	frame/sec
512 kbps	320x240	low	25
1024 kbps	320x240	high	25
1536 kbps	640x480	low	25
3072 kbps	640x480	high	25

A. Scalability Layers

The first step for the integration of scalable content into the NextShare system was the selection of the desired scalability layers. The selected layers are described in Table I.

As illustrated in Table I, four scalable layers were selected for the integration. The main reasons for selecting this layer structure were to maintain a good coding efficiency and to provide all popular qualities. The possibility to add further layers to support HD content is also fully supported by our framework, but has been omitted for the current version due to constraints in the upload bandwidth of our system's users. From the coding-efficiency point of view, the difference between the layers in terms of bit rate should be not too low, as the coding efficiency decreases drastically in such cases [14], while the selected bit rates represent the most popular qualities that are provided nowadays by multimedia portals. Furthermore, it should be noted that the audio bitstream is provided together with the video bitstream of the base layer. Thus, the 512 Kbps for the base layer includes the bit rate for the 128 Kbps audio bitstream. This is necessary to ensure that the audio is always received in time for playback, which can start as soon as the base layer is received.

To ingest the different layers into the P2P system the layers need to be provided as separated files. The base layer is multiplexed with the audio content and provided in a proper container format. The enhancement layers are provided as separate optional files. By using this file structure, Bittorrent clients without SVC support can still download the H.264/AVC-compatible base layer and decide not to download the optional enhancement layers without wasting any bandwidth.

B. Mapping to Bittorrent Pieces

The second step of the integration process is the mapping of the scalability layers to Bittorrent pieces. Firstly, the unit shall represent a synchronization point for dynamic switches between different quality layers. To achieve this goal each unit starts with an Instantaneous Decoding Refresh (IDR) reference frame. Secondly, it should be noted that we do not perform a direct mapping to pieces but to a unit. This unit represents a fixed number of frames for a specific layer and can be mapped to a fixed number of pieces. The reason for this approach is that the piece size might be changed in the P2P system for various reasons, and by basing the mapping

Table II
UNIT MAPPING

Layer	Kb/time slot	KB/time slot	pieces/time slot
BL	$512Kbps * 2.56 \approx 1.310$	$/8 \approx 164KByte$	3 pieces @ 55 KByte/time slot
EL1	$1024Kbps * 2.56 \approx 2.621$	$/8 \approx 328KByte$	6 pieces @ 55 KByte/time slot (3 pieces in previous layers, 3 new pieces)
EL2	$1536Kbps * 2.56 \approx 3.932$	$/8 \approx 492KByte$	9 pieces @ 55 KByte/time slot (6 pieces in previous layers, 3 new pieces)
EL3	$3072Kbps * 2.56 \approx 7.864$	$/8 \approx 983KByte$	18 pieces @ 55 KByte/time slot (9 pieces in previous layers, 9 new pieces)

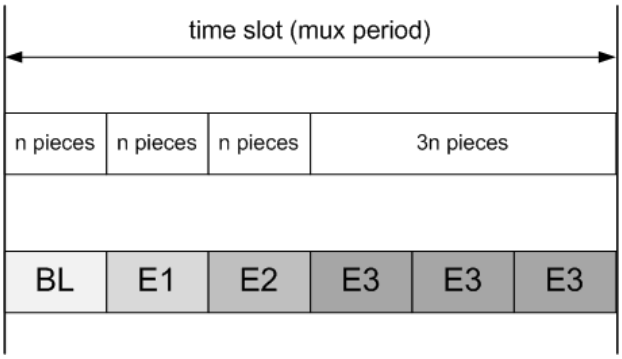

Figure 1. Piece Mapping

on units rather than on pieces only the unit/piece-mapping needs to be updated when the piece size is modified.

The mapping to the units has been performed based on several criteria. First, the units need to be selected large enough to allow for a good coding efficiency. As it should be possible to decode each unit independently (when all lower layer units for the same time stamp are also available) the number of frames within one unit should be high enough to allow for good coding efficiency. Additionally, the number of frames within one unit should be low enough to provide the flexibility to conveniently switch between qualities when the network conditions change.

Based on these considerations, a mapping of 64 frames, which represent 2.56 seconds of content at a frame rate of 25 frames/sec, has been selected. Such a unit is subsequently mapped to three pieces; however, as noted previously, the piece mapping can always be changed based on the requirements from the P2P system. The piece mapping is illustrated in Table II.

The mapping to the 55 KByte pieces results in a small overhead of available bits per piece. However, this overhead is utilized to compensate the small drifts of the constant bit rate (CBR) algorithm utilized during the SVC encoding process (see Section IV-A).

Based on the calculations in Table II, a mapping of the layers to Bittorrent pieces could be performed as illustrated in Figure 1.

The figure shows that the unit for each layer can be mapped to a specific number of actual pieces.

IV. PRODUCER-SITE ARCHITECTURE

The producer-site architecture describes all steps from encoding the SVC bitstream to the ingestion into the core of the P2P system. The topics addressed in this section include the encoding process, the splitting of the bitstream, creating metadata based on the bitstream's supplemental enhancement information (SEI), packetizing the bitstream, and ingesting the bitstream into the core of the P2P system. An illustration of this architecture is provided in Figure 2, more details on each of the processing steps are provided in the following sections.

A. Bitstream Preparation

As the first step of the bitstream preparation process, the raw video (i.e., the YUV video frames) is encoded by an optimized JSVM 9.15 [4] encoder, which uses a CBR algorithm to ensure that the pieces created from the video content have a constant size. The CBR algorithm works at GOP (Group of pictures) level and maintains the bit rate at GOP level throughout the encoded bitstream. However, the CBR algorithm still produces a small offset compared to the desired bit rate. As a constant piece size has to be maintained, a positive offset could results in frame dropping while a negative offset can be easily addressed by using padding bits during the splitting process. To ensure that no frames are dropped in case the small drifts of the CBR algorithm result in a positive offset, the target bit rate for the CBR algorithm is chosen slightly lower (approx. 1-2 % below the target bit rate). Thus, the CBR algorithm produces only negative offset compared to the real target bit rate, which can be easily handled.

The encoded SVC bitstream is subsequently split into the H.264/AVC-compatible base layer (BL) and the enhancement layers (EL) by the Network Abstraction Layer Unit (NALU) demuxer. The demuxer analyzes the NALU headers and splits the access units into separate bitstreams for each layer. Each of these layer bitstreams consists of several pieces of constant size. If within one bitstream the GOP size exceeds the piece size, subsequent NALUs (frames) would be dropped. However, as mentioned in the previous paragraph, such a situation is avoided by setting a slightly lower target bit rate for the CBR algorithm. If the GOP size is less than the piece size, the remaining size bits are filled with padding bits. Additionally, the SEI information at the beginning of the bitstream (i.e., the scalability info message) and the Sequence Parameter Set (SPS) and Picture Parameter Set (PPS) are provided to the metadata creator (see Section IV-C).

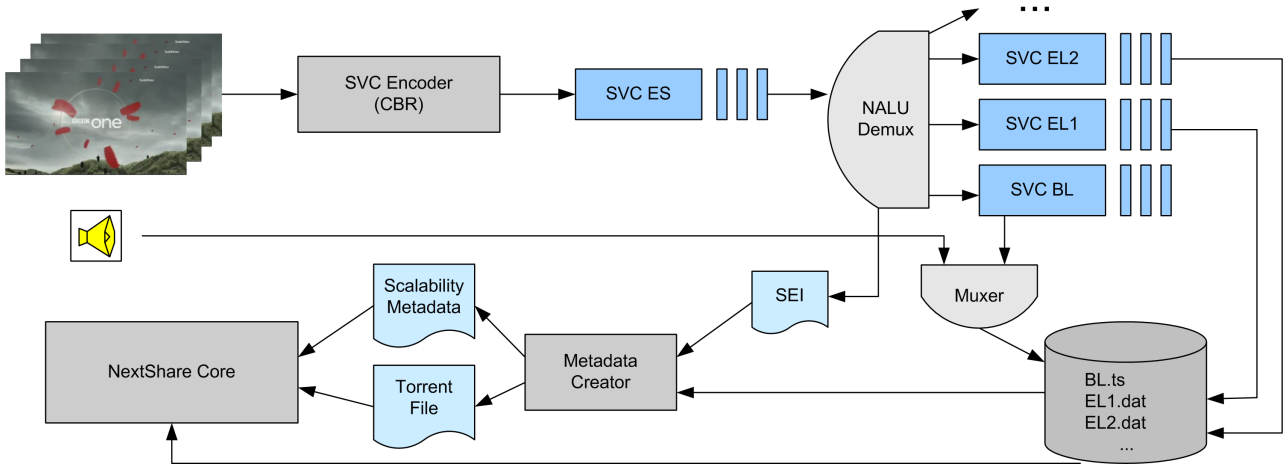

Figure 2. Producer-Side Architecture

The audio data can be provided already encoded, e.g., as an MP3 or AAC audio file. If a raw PCM audio file is provided, the audio content is encoded to the desired audio coding format.

B. Bitstream Packetizing

In the bitstream packetizing step, the base layer of the SVC bitstream is muxed with the audio into a proper container format. The main reason for this step is that the base layer should be provided in a backwards-compatible way, so that also end user P2P clients or terminals that only support H.264/AVC can successfully process the base layer. For such a purpose two different container formats were investigated for our system, the MPEG Transport Stream (MPEG-TS) [5] and the MPEG-4 file format (MP4) [6].

MPEG-TS is a standard able to encapsulate audio and video Packetized Elementary Streams (PESs) and other data and is supported by a majority of systems and applications. The main disadvantage in using MPEG-TS is that it usually has a rather high overhead in terms of bit rate(10-20% in average). An alternative muxing scheme is provided by the MP4 format which provides functionalities similar to the ones of MPEG-TS while having a clearly lower overhead ($\sim$1%). Thus, MP4 is the preferred container format used in our system, while MPEG-TS support is provided for compatibility to older systems.

The overall architecture is codec-independent: the system is able to recognize the container format and apply the corresponding processing. A general problem during the muxing phase is that the output should have a certain fixed size to ensure that a full GOP of video content and the corresponding audio content can be mapped to one unit. Considering that muxing schemes can have variable overheads, it is in principle not possible to a priori know if the output of the muxer for a certain audio and video input will respect the size limits. In case the output size is smaller than expected it will be possible to add padding bits and solve the issue (muxing codecs usually provide routines for that). The real problem is when the muxing output size is higher than the allowed one: in such a case the muxer tries to change its parameters to lower the overhead to the minimum. However, if adjusting the muxer's parameters is not sufficient, it would usually not be possible to meet the size constraints. Thus, as previously mentioned in Section IV-A, the target bit rate is set lower than desired to ensure that only the first case (lower output size) occurs. To avoid possibly wasting too much bit rate on padding bits, the architecture optionally provides support for a feedback mechanisms between the muxer and the encoders to solve this. Thus, in case the output size would be higher than the target size, the muxer asks the SVC and the audio encoders to re-encode both audio and video using a lower target bit rate. For the enhancement layers, the padding mechanism described in the previous section is applied.

C. Scalability Metadata Support

Although the pieces of the video stream are transmitted over the network in a layered way, the de-packetizer at the consumer-site needs to know the properties of the layers for the decoding process and the decoder needs access to the parameters from the beginning of the bitstream (the SPS and PPS elements). Thus, the properties of the layers, which are usually provided by the Supplemental Enhancement Information (SEI) at the beginning of the bitstream, and the parameter sets need to be forwarded to the consumer-site.

To store these metadata and transmit them to the de-packetizer when needed, the SEI message and the parameters are forwarded from the NALU demuxer to the metadata

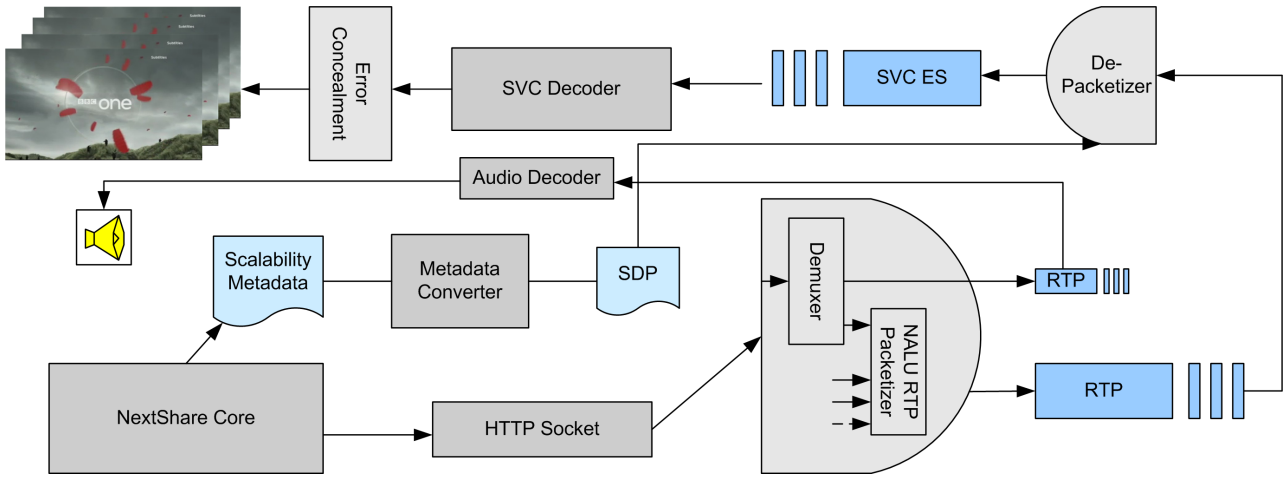

Figure 3. Consumer-Side Architecture

creator. The metadata creator subsequently parses the SEI data and stores the properties of the layers in an XML metadata document. Additionally, the SPS and PPS elements are encoded in base64 to allow their storage in XML and are added to the metadata document. The resulting metadata document contains all the layer information and parameter sets required by the de-packetizer and decoder modules (see Section V for details).

D. Ingest into the Core

The NextShare core represents the P2P engine responsible for creating and injecting the content into the network. The main metadata file required for the ingestion of the content into the P2P system is the torrent file. The torrent file provides the information required for the download of the previously encoded base and enhancement layers, as well as metadata related to the content including the previously created scalability metadata. The created torrent file is compatible with the Bittorrent protocol [1] and can therefore be processed by every peer running a Bittorrent-compatible client. This backwards-compatibility increases the reliability of the torrent swarm and the scalability of the distribution costs, as the torrent file can not only be processed by NextShare-compatible clients, but by all Bittorrent-compatible clients.

The fact that the H.264/AVC-compatible base layer and the audio stream are provided commonly packetized into a proper standardized muxing format enables also clients without SVC support to join the swarm as they are able to consume the stream in base layer quality. Therefore, every peer has an incentive to download at least the base layer, which increases its availability in the swarm. As the base layer is the most important layer (it is sufficient to start the playback and is always required) this really helps to ensure real-time playback for all clients in the swarm.

After the creation of the torrent file, the content is ingested into the NextShare core (i.e., the torrent file is distributed to other peers and the files containing the base- and enhancement layers are seeded). During the ingestion process, the base and enhancement layer files are split into pieces as illustrated in Figure 1. During the mapping process some problems have to be taken into consideration. Firstly, the integrated CBR algorithm does not provide an exactly constant bit rate, but allows for minor drifts. Thus, the piece mapping is always performed with a small overhead. Additionally, the 188 byte size utilized for the MPEG-TS packets cannot be mapped to a power of two, while the pieces ingested into the NextShare core should have a multiple of two as their size. Thus, the possibility of choosing a piece size that differs from the standard power of two has been successfully investigated.

V. Consumer-Site Architecture

The consumer-site architecture describes all steps from receiving the bitstream through the P2P system to decoding the bitstream for displaying it in the media player. The steps described include the local streaming of the received content, the de-packetizing of the content, the signaling of the layer properties using suitable metadata, and the merging and decoding of the received layers. An overview of this architecture is provided in Figure 3 and described in detail in the subsequent sections.

A. Provision of the Received Content

When the content is accessed by the user the layers from the NextShare swarm are downloaded in an intelligent way in order to maximize the Quality of Experience (QoE) for the current available download bandwidth. A great advantage of changing quality by displaying more or less enhancement layers regards the fall-back scenario: in a P2P system peers

are considered to have an unreliable and selfish nature, leaving the swarm and decreasing the total available bandwidth as soon as they have received the desired content.

In such a case the fall-back scenario will occur, where the user will experience a slow decrease of the QoE, as the download engine avoids downloading higher layers for upcoming time slots, if enough pieces of the previous layers are not already available. Retrieving the content from the network is based on a modified approach of the Give-to-Get (G2G) algorithm [11]. The main reason for using G2G is that Bittorrent's Tit-for-Tat algorithm is not suitable for streaming multimedia content.

The original G2G algorithm divides the part of the piece buffer close to the current playback position into high-, mid-, and low-priority sets with regard to the current playback position. The high-priority set is the part following immediately after the playback position. The G2G algorithm selects the pieces in the high-priority set based on their deadline, i.e., the piece with the nearest deadline in the high-priority set is downloaded first to ensure a continuous playback of the content. In the mid- and low-priority set, the pieces are selected using Bittorrent's rarest first strategy. Using this piece-picking policy the G2G algorithm tries to ensure that the pieces are downloaded in time for playback while still ensuring that piece's that are desired by neighbour peers are downloaded as well.

For the layered application of the G2G algorithm, the priority sets are applied to all the active available layers in a proportional way. Thus, for every layer a high-priority set is created where the pieces are selected according to their deadline while the dependency between the layers is considered.

As discussed in the previous section, by providing backwards compatibility with other clients, the availability of the base layer at all peers in the same swarm can be assumed. Therefore, if a download bandwidth of at least the base layer's bit rate is provided by the peer's connection, it can also be assumed that the playback will never stall (if the neighbour peer seeds the content or is ahead in its playback position). Note that once a peer finishes watching the content, the download engine will start retrieving the remaining pieces of all the layers for two major reasons: firstly, to increase the layer's availability in the swarm, and secondly, to enable watching the content at the highest quality again once all the layers have been downloaded. However, if this behaviour is not desired due to bandwidth restrictions it can be disabled in the configuration.

After enough content has been downloaded to guarantee a continuous playback of at least the base layer, the download engine of the NextShare core will initialize the demuxer module. The multimedia data is forwarded to the demuxer utilizing an HTTP socket, which was selected to ensure interoperability between the NextShare core and third-party de-packetizing/decoding solutions. A persistent connection will be established between the demuxer and the NextShare core, allowing the demuxer module to sequentially ask for the available content for the following time slot, depending on the requests it receives from the decoder module. It is important to notice that the available pieces of the following time slot will be sent to the demuxer as late as possible, allowing the NextShare core to manage the major buffer, to increase the quality until the last moment and to try to increase the quality if the user pauses the playback.

B. Bitstream Demuxing and Packetizing

As described before, the demuxer works online: it receives from the swarm at least the stream of the base layer, which is processed by a suitable demuxer. The demuxer firstly removes any possible padding from the production phase and splits the container format stream into the audio content and the H.264/AVC-compatible base layer.

The elementary audio stream is directly encapsulated into a Real-Time Transport Protocol (RTP) packet stream (e.g., according to [9] for an MP3 audio stream) and is sent to the successive module. The elementary SVC base layer video stream is forwarded to the NALU RTP packetizer, which puts the base layer and the received enhancements layers into an RTP packet stream, reordering the packets and forwarding the RTP stream containing all layers to the next module. Furthermore, the demuxer establishes a RTCP channel with the player. This is useful in order to maintain essential synchronization information among the video and the audio layers and also to support playback control commands.

Please note that all the modules represented in Figure 3 are typically running on the same host, i.e., the peer that receives the content. However, the main reason for selecting the RTP protocol to convey the audio and video data was to provide flexibility and to enable a possible integration of the P2P network with a more traditional server-based network. In such a server-based network the NextShare consumer-peer could act additionally as a server, receiving the content from the P2P network and redistributing the scalable content within an RTP streaming network.

C. Scalability Metadata Support

To decode the layers received from the NextShare system, the de-packetizer needs to be aware of the scalability properties of these layers. To provide a generic signaling mechanism for these properties, which could also be used by third-party solutions, we have decided to provide this information as a Session Description Protocol (SDP) document to the de-packetizer. The SDP document is formatted according to [13], which provides the capabilities to signal the properties and dependencies of the scalable layers.

As the metadata is provided by the NextShare core in a NextShare-conforming XML format, the scalability metadata document is firstly converted to the targeted SDP format. Subsequently, the SDP message containing the layer

A) Correctly received EL3 picture: PSNR 42,4 dB

B) Upscaled picture from the base layer: PSNR 31,7 dB

Figure 4. Upscaling Result

properties and the parameters sets is forwarded to the de-packetizer.

D. Bitstream Consumption

After the demuxing and RTP packetizing of the audio and video content, the RTP streams are forwarded. While the audio stream is forwarded to a standard de-packetizer and decoder, the SVC RTP stream is processed by our customized tools.

Firstly, our SVC de-packetizer parses the RTP packets and provides the payload and the time stamps to the SVC decoder. To perform this extraction process, the de-packetizer needs to have SVC layers properties along with the audio and video RTP port information in advance. This information is provided by the SDP file. Additionally, the de-packetizer parses the parameter sets from the SDP and provides them to the decoder, as they are required for the decoding process. The SDP file contains the SVC layer information for each layer and the de-packetizer extracts the suitable information for the desired layer playback. Finally, our highly optimized version of the JSVM 9.15 reference decoder performs the real-time decoding of the SVC content utilizing the error concealment algorithm embedded in the decoder (described in the next section).

E. Error Handling

Error robustness in the video decoder is important since transmission errors are very common in current (especially wireless) video streaming and transmission systems [16]. The transmission errors can lead to very poor quality of experience and in worst case scenario, they can lead to decoder crashes. Usually the error concealment is performed by monitoring the order of the NAL slices and their macroblocks to see if all the NALUs are received. If NALUs are missing suitable concealment operations for the missing macroblocks are performed, e.g., by using a frame copy from the previously correctly received frame [17].

The difference in the error concealment implementation in the NextShare system to such traditional error concealment approaches is that random frames or burst of frames cannot occur, as each piece contains a full GOP for a specific layer. Thus, a GOP can either be received or not received, but single frames cannot be lost during the transmission. As the whole GOP between the IDR pictures can either be present or missing, this can lead to varying resolution in the player if spatial scalability layers exist (as suggested in Table I). However, a major advantage of the layered content provisioning in NextShare is the awareness of the layer dependency. The higher enhancement GOPs are not send to the decoder if not all the pieces from the lower layer GOPs have been previously sent for the current time slot. Additionally, the base layer is always retrieved, which makes the error concealment easier and more effective.

The error concealment is integrated into the optimized JSVM 9.15, which was integrated into the VLC plug-in. To cope with the missing spatial enhancement layers, an upscaling functionality was integrated into the decoder based on normative integer-based 4-tap filters. The upscaling algorithm is provided by the JSVM reference software; please refer to [4] for more details.

The target resolution for the sequence is defined in the SPS NAL unit, which is compared with the received resolution (the resolution of the IDR picture) when starting the

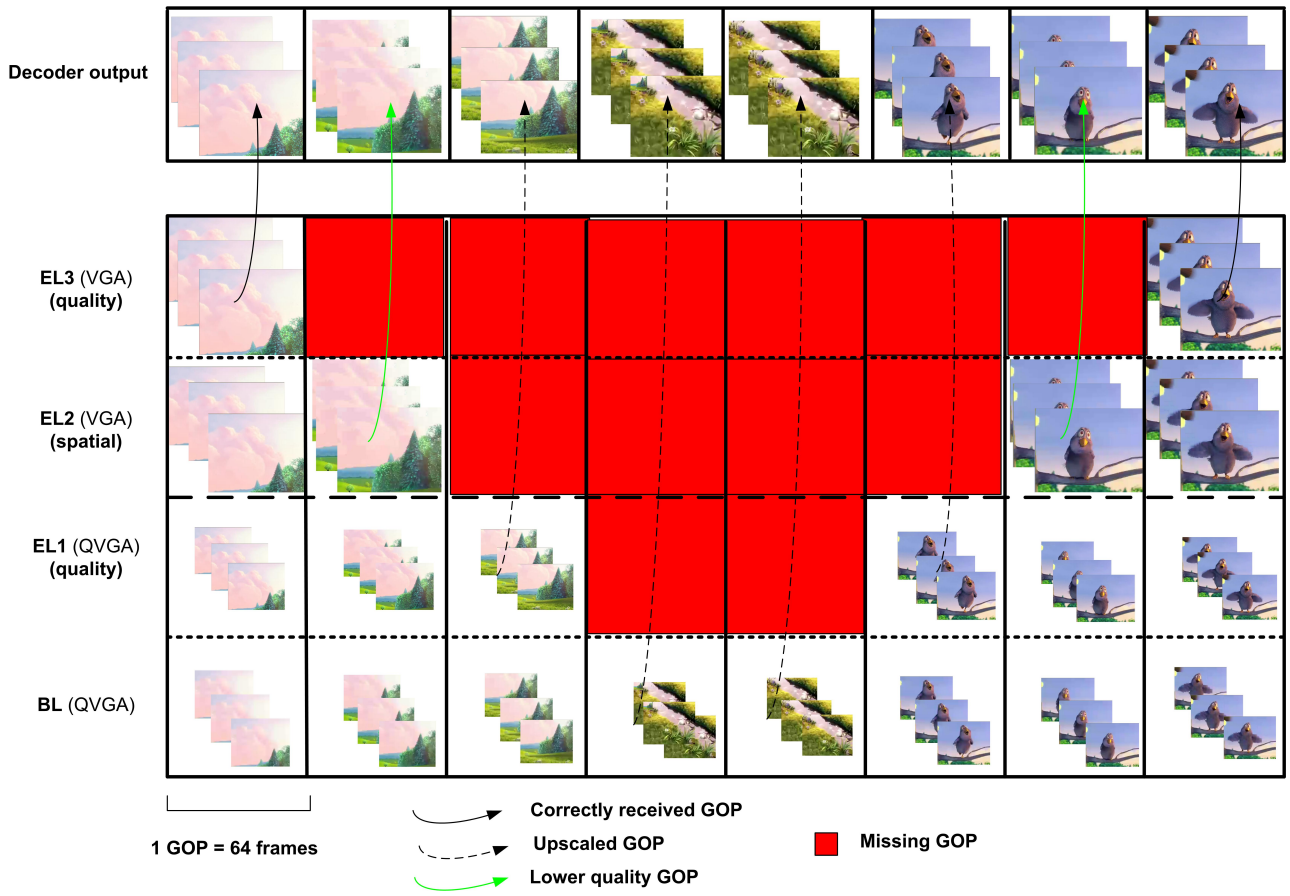

Figure 5. Upscaling Principle

decoding for a new time slot. As mentioned earlier, the SPS information reaches the decoder within an SDP description from the RTP de-packetizer. If the resolutions do not match, frames of the new time slot are up-scaled from the lower resolution to the target resolution to maintain the preferred resolution. Since the layer with lower spatial resolution usually provides lower quality, the upscaled picture is blurred, as illustrated in Figure 4. However, compared to changing the window size during video playback, the blurring is usually the better solution.

The selected structure of layers presented in Table I supports two different resolutions for the video. In some cases the layers for high resolution video cannot be received within the defined time slot, even though the consumer prefers to watch the video in higher resolution. As an example, this could happen while streaming on a heavily congested access point. Figure 5 shows an example situation with a four layer SVC sequence, where the second enhancement layer provides the spatial enhancements whereas the first and third enhancement layers provide quality enhancements. In Figure 5, the number of received layers decreases during the streaming from best quality to the base layer level. In this case, an upscaling of the base layer quality to the higher resolution is performed, to avoid changing the playback window size, which is very disturbing for the consumer. The upscaling algorithm is performed when only the base layer or the base and first enhancement layer are received. As soon as the second enhancement layer is received, no upscaling is necessary as the desired resolution is already provided.

VI. EVALUATION

To investigate the performances of our solution we performed a series of experiments with the implementation of our architecture. The experiments were performed in a lab-only environment composed of several peers connected with each other using heterogeneous connections. We monitored the performance of a peer acting as leecher in the swarm, retrieving the content from at least one seeder. Each of the seeders provides a limited upload bandwidth capacity to the leeching peer, which limits the download bit rate the leecher can achieve.

For our tests we encoded a two minute video sequence using four quality layers. The properties of these layers are

Table III
EVALUATION LAYER STRUCTURE

Bit Rate	Resolution	Quality	frame/sec
512 kbps	640x480	basic	25
1024 kbps	640x480	low	25
1536 kbps	640x480	medium	25
2048 kbps	640x480	high	25

described in Table III. The properties of the sequence used for the evaluation differ slightly from our reference layer structure described in Table I. The main reason for using a different layer structure for the evaluation were to keep a simple uniform unit size for all layers. Additionally, only quality layers were used for this evaluation to enable an easy comparison of the received quality layers in terms of peak signal-to-noise ratio (PSNR).

The evaluation consists of two test series. First, the behaviour of the layered piece download algorithm is illustrated in different scenarios to demonstrate the efficiency of our layered piece-picking implementation (Section VI-A). Second, the received quality of the layered implementation is compared to the single layer implementation of our P2P system to show how improvements in terms of received quality can be achieved (Section VI-B).

For the evaluation process five scenarios for the two minute test video sequence were defined. Four of those scenarios were investigated for both test series, while the first scenario is only interesting for the layered piece-picking series. In the first three scenarios all peers remain in the swarm for the whole time. In the other two scenarios, seeders leave the swarm at specific time points to test the robustness of the system against churn.

- Scenario 1: The leecher peer connects to a single seeder, which provides sufficient bandwidth to download only the base layer. This scenario is only investigated for the first test series.
- Scenario 2: The leecher peer connects to three seeding peers. Together the seeders provide enough bandwidth to download all layers for the test video.
- Scenario 3: The leecher peer connects to two seeders, which provide more than sufficient bandwidth for the download of three layers, but not sufficient to constantly download all layers.
- Scenario 4: At the beginning of this scenario two seeders provide sufficient bandwidth to download all layers. After 40 seconds one of the seeders leaves the swarm and the available bandwidth is decreased.
- Scenario 5: At the beginning of this scenario three seeding peers are in the swarm and sufficient bandwidth for all layers is provided. After 30 seconds, one seeder leaves the swarm and decreases the available bandwidth. After 70 seconds, a new seeder joins the swarm and increases the bandwidth.

As the player of our system can switch between qualities without flickering (see Section V-E) for this evaluation the playback quality is increased as soon as possible, even if only for one time slot. This particular behaviour can be changed to a more conservative approach that might be needed in case the decoder/player module is replaced or a constant quality playback is preferred. To influence the download strategy, the piece-picking algorithm can be configured to only perform switches to higher layers if a specific number of higher layer pieces have been downloaded (this number is one for the presented evaluation, i.e., immediate quality switches are performed).

A. Layered Piece-Picking Test Series

In the following graphs, representing the first test series, all five scenarios are presented to analyze the behaviour of the piece selection when layered content is streamed. In every graph the available upload capacity, the download rate, and the received bit rate are presented. The upload capacity illustrates the download bandwidth that is provided from the seeders to the leecher peer. The download bit rate describes at which rate the leecher peer downloads pieces from the seeders. Please note that the download speed is never calculated instantly, instead it is based on the piece arrivals and therefore averaged over a small period of time. This explains the smoothness of the download curve and avoids spikes in the results. Finally, the received bit rate represents the number of pieces that were received in time for playback. It should be noted that the received bit rate differs slightly from the actual playback bit rate of the video. The reason for this deviation is that the CBR algorithm of our system's SVC encoder does not provide an exactly constant bit rate but allows for small drifts. Thus, padding is used to achieve the constant piece size, as described in Section IV-A.

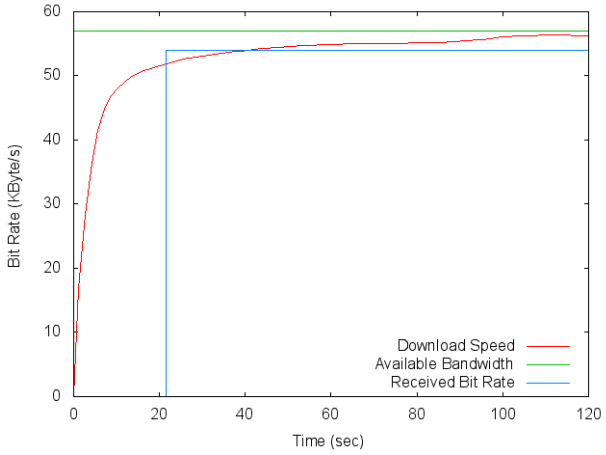

Figure 6. Layered Piece-Picking Series: Scenario 1

In Figure 6 the results for the first scenario are presented. As the available download bandwidth allows to download

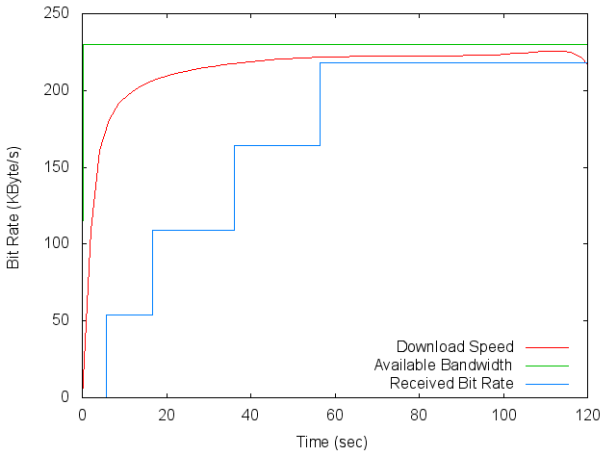

Figure 7. Layered Piece-Picking Series: Scenario 2

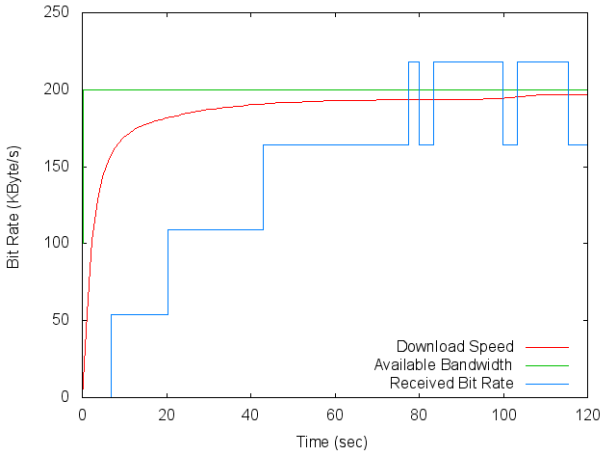

Figure 8. Layered Piece-Picking Series: Scenario 3

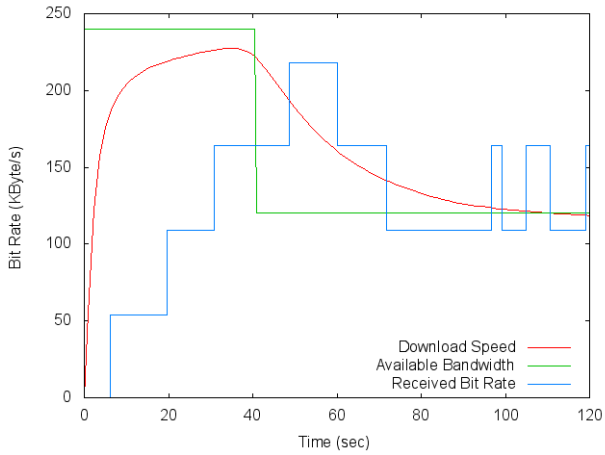

Figure 9. Layered Piece-Picking Series: Scenario 4

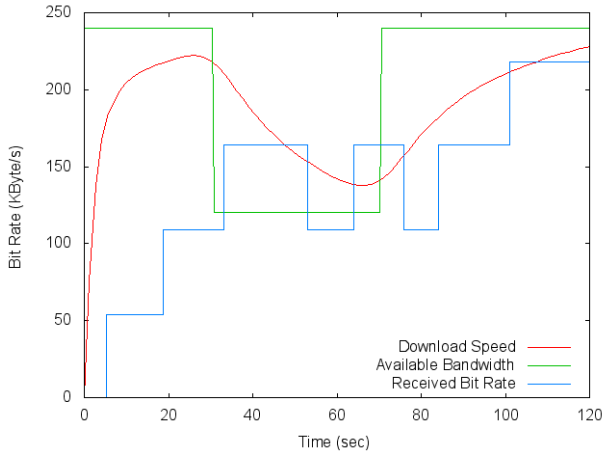

Figure 10. Layered Piece-Picking Series: Scenario 5

only the base layer, the behaviour of the piece-picking algorithm is very simple. Only base layer pieces are downloaded and the playback of the base layer is started after the initialization phase.

Figure 7 illustrates the behaviour in case of the second scenario. As mentioned before, a switch to a higher layer is performed as soon as the first piece of the layer is downloaded. Thus, the leecher peer starts with the playback of the base layer and gradually increases the quality until the highest layer is reached. The playback remains at best quality until the end of the scenario.

An interesting behaviour emerging from the configuration of the piece-picking algorithm for this test series can be seen in Figure 8. Having a download rate higher than the first three layers combined, the piece-picking algorithm will try to download the highest layer whenever possible. As the available bandwidth is not sufficient to constantly download all layers, the playback quality switches frequently. If a constant playback quality is preferred, the piece-picking algorithm can be configured accordingly.

In Figure 9 the fourth scenario is presented. At the beginning of the scenario the playback quality is gradually increased, similar to the behaviour in the second scenario. However, as soon as the highest playback quality is achieved, one seeder leaves the swarm and causes a decrease of the available bandwidth. Thus, the leecher peer cannot download the higher layer pieces anymore and decreases the playback quality. Nevertheless, the playback never stops and the piece-picking algorithm stabilizes after some time and downloads the best possible quality for the given bandwidth until the end of the scenario (between two and three layers, similar to the behaviour in scenario 3 but with less available bandwidth). The results of this test scenario show that our implementation and is robust to departing peers and can keep the video playback going, as long as sufficient seeding peers to download at least the base layer quality remain in the swarm.

Figure 10 illustrates the results for the final scenario. At

the beginning the playback is again gradually increased. However, one of the seeding peers leaves the swarm after 30 seconds and the piece-picking algorithm adjusts and downloads the best possible quality for the new bandwidth conditions (between two and three layers). After the joining of the new seeder, the playback quality is increased to the best possible quality, which can now be downloaded due to the improved bandwidth conditions. The results of this scenario show that our system is robust to churn and reacts suitably to leaving and joining seeders.

Another interesting observation of this test series is the initial playback delay when streaming layered content using our NextShare system. The test results of all scenarios show that the playback will start no more than ∼25 seconds after the leeching peer joins the swarm, assuming a download rate at least as high as the base layer's bit rate (otherwise no real-time playback is possible). Additionally, the playback of the base layer can start as fast as after five seconds, if the bandwidth conditions are good. This reduces the initial playback delay greatly compared to the single layer implementation of our system.

B. Quality Comparison Test Series

In this test series a comparison of the single layer and multi layer implementations of our system is performed. The goal of this test series is to compare the received quality for both approaches. For the evaluation process the previously described scenarios 2-5 are investigated.

The following graphs represent the results for this test series. Each graph illustrates the received video quality in PSNR for the single and multi layer implementations. The PSNR for a received piece is calculated by averaging the PSNR of all 64 frames contained within a piece for the multi layer implementation (the piece structure is described in Section III). For the single layer approach, one piece contains in average 16 frames providing similar quality as all four layers of the multi layer approach (the same piece size is used for both implementations). The received PSNR is calculated by summating the PSNR of the frames in the received pieces and dividing it by the total number of frames for the time slot.

In Figure 11 the results for the second scenario are displayed. During the initial phase where the quality is gradually increased, the received quality is clearly better for the multi layer implementation, as the base layer already provides a decent quality, while the single layer implementation only receives a part of the frames. However, when the highest quality playback is achieved (download of all pieces), the single layer approach shows slightly better results. This is due to the fact that a better PSNR can be achieved with single layer encoding at the same bit rate, as the multi layer encoding has a certain overhead in terms of coding efficiency (8.6 % for the example test sequence).

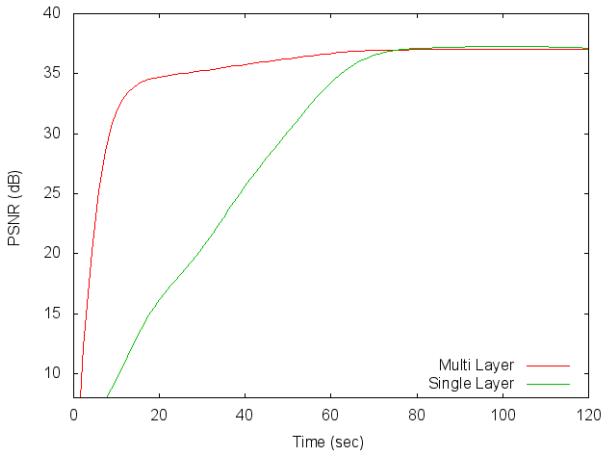

Figure 11. Quality Comparison Series: Scenario 2

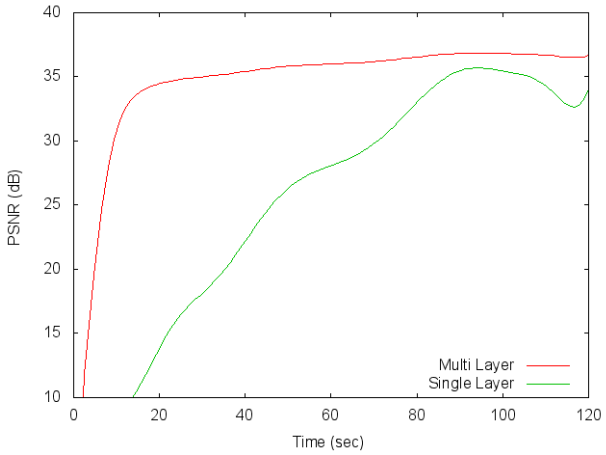

Figure 12. Quality Comparison Series: Scenario 3

Figure 12 shows the results for the third scenario. As the available bandwidth is not sufficient to constantly download all pieces, the single layer implementation has always a lower average PSNR than the multi layer implementation.

The results of scenario four in Figure 13 are similar. As one of the seeders leaves the swarm after 40 seconds, the average PSNR for the single layer approach remains rather low, while the multi layer approach can fall back to the lower layers and in comparison does not lose much in terms of average PSNR.

In Figure 14 the results for the fifth scenario are illustrated. Due to the leaving of one of the seeders after 30 seconds the average received quality of the single layer approach stays rather low. However, after the joining of the new seeder the quality for both implementations is increased and at the end of the scenario the quality for the single layer approach is slightly better, as all pieces are received in time.

Overall, the multi layer implementation has shown clearly better performance in terms of received quality during this

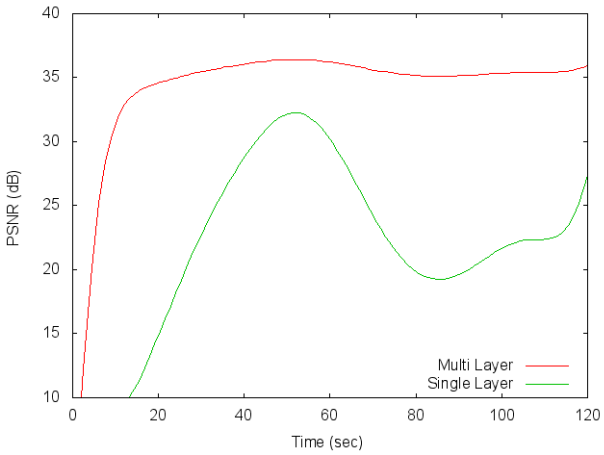

Figure 13. Quality Comparison Series: Scenario 4

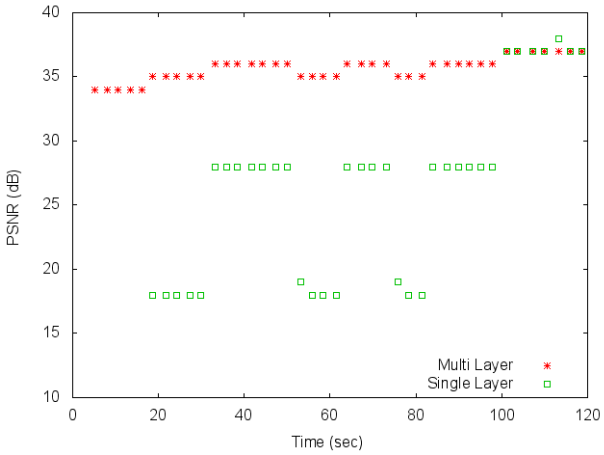

Figure 14. Quality Comparison Series: Scenario 5

test series. The single layer approach can only provide better quality if there is always more than sufficient bandwidth to download all pieces in time and no fluctuations occur. Additionally, for this evaluation we have only considered rather high bandwidth scenarios where most of the pieces can be downloaded in time. In low bandwidth scenarios the performance of the single layer approach gets even worse compared to the multi layer approach (only a smaller part of the frames is received in time).

VII. Future Work

Although our framework already provides a full solution for the streaming of scalable content over P2P networks, there are still open possible modifications that could enhance the viewing experience for the user.

Firstly, the current approach only utilizes the layered scalability provided by the SVC standard, i.e., the scalability in terms of resolution and fidelity. As a further step an additional support of temporal scalability would be desirable. Such a support could be realized by reordering the frames according to the hierarchical prediction structure and by storing the frames for different temporal levels in different pieces. However, this would increase the complexity of the piece-picking process and the trade-off between the increased complexity and the enhanced viewing experience needs to be investigated.

Another desirable further step would be the support of medium-grain scalability. However, such a support would require some changes to our architecture. As the piece picking algorithm currently works on piece level, where each piece contains a number of NALUs, and the medium-grain scalability allows to change the quality on NALU level, a new algorithm has to be designed to allow the partial download of pieces (i.e., to work on chunk level, with the need to investigate compatibility with existing clients). The other possibility would be to switch to dynamic piece size and map each NALU to a different piece. However, such a small piece such would be a bad choice with regard to the overhead of the piece sharing and a dynamic piece size would again break the compatibility with existing clients.

Furthermore, additional evaluations of our system will be performed to test the system's performance in large scale trials of our project's living lab [2]. Additionally, different configurations of the piece-picking algorithm will be evaluated to determine in which situations which piece-picking and quality switching strategies are preferable.

VIII. Conclusion

In this paper a framework for the provision of scalable content in a fully distributed P2P system was presented. In Section II, the selection of the scalable layers was explained and the mapping of the layers to units and subsequently Bittorrent pieces was illustrated. The main reasons for the choice of 64 frames per unit were to achieve a good coding efficiency while still maintaining the flexibility to quickly switch between layers if the network conditions change. In Sections III and IV the producer- and consumer-site architecture were presented. The producer-site architecture includes the encoding of the SVC bitstream and splitting of the bitstream into layers, the packetizing of the base layer and the audio stream into a suitable container format, the creation of the scalability metadata based on the SEI information at the beginning of the bitstream, and the ingest of the content into the core of the P2P system. On the consumer-site, the modules for retrieving and consuming the content were presented. The retrieved content is provided to the demuxer utilizing an HTTP socket and the demuxed audio and video streams are forwarded to the media consumption solution using RTP. The de-packetizing and decoding of the SVC content is subsequently performed by our customized SVC tools utilizing the scalability information provided by a suitable SDP document.

In the evaluation section, the performance of our P2P system supporting scalable content was evaluated. Two test series were performed in order to evaluate the performance of our piece-picking algorithm and to compare the new multi layer implementation to the already existing single layer implementation in our system. The results of the first test series show that the layered piece-picking algorithm can efficiently utilize the available bandwidth after the initialization phase. Additionally, the layered implementation greatly reduces the start-up delay and is robust to churn. The second test series showed that the multi layer implementation shows a clearly better performance than the single layer implementation in all situations where the bandwidths is restricted or bandwidth fluctuations occur.

The presented system is to our knowledge the first open-source P2P system with full SVC support. Additionally, it is fully compatible with third-party media consumption solutions due to utilizing the HTTP, RTP, and SDP protocols for providing the content from the P2P system to the media consumption modules. Thus, the system is easy to use and customize for interested users.

ACKNOWLEDGMENT

This work is supported in part by the European Commission in the context of the P2P-Next project (FP7-ICT-216217) [3].

REFERENCES

[1] Bittorrent protocol 1.0. URL: http://www.bittorrent.org. Last accessed 15-July-2011.

[2] The P2P-Next living lab. URL: http://livinglab.eu. Last accessed 15-July-2011.

[3] The P2P-Next project. URL: http://www.p2p-next.org. Last accessed 15-July-2011.

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

[5] ISO/IEC 13818-1. *Generic coding of moving pictures and associated audio information: Systems*, 2000.

[6] ISO/IEC 14496-1. *MPEG-4 Part 14: MP4 File Format Version 2*, 2003.

[7] N. Capovilla, M. Eberhard, S. Mignanti, R. Petrocco, and J. Vehkapera. An architecture for distributing scalable content over peer-to-peer networks. In *Second International Conferences on Advances in Multimedia (MMEDIA)*, pages 1–6, 2010.

[8] P. Baccichet et al. Low-delay peer-to-peer streaming using scalable video coding. In *Packet Video*, pages 173–181, 2007.

[9] R. Finlayson. A more loss-tolerant RTP payload format for MP3 audio. RCF 3119, 2001.

[10] Z. Liu, Y. Shen, K. W. Ross, S. S. Panwar, and Y. Wang. Layerp2p: Using layered video chunks in P2P live streaming. *IEEE Transactions on Multimedia*, 11(7):1340–1352, August 2009.

[11] J. J. D. Mol, J. A. Pouwelse, M. Meulpolder, D. H. J. Epema, and H. J. Sips. Give-to-Get: Free-riding-resilient video-on-demand in P2P systems. In *Multimedia Computing and Networking*, volume 6818, San Jose, USA, 2008.

[12] R. Rejaie and A. Ortega. Pals: Peer-to-peer adaptive layered streaming. In *NOSSDAV Proc.*, pages 153–161, New York, NY, USA, 2003.

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

[14] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, September 2007.

[15] A. Sentinelli, L. Celetto, D. Lefol, C. Palazzi, G. Pau, T. Zahariadis, and A. Jari. A survey on P2P overlay streaming clients. In *Towards the Future Internet - A European Research Perspective*, pages 273–282, 2009.

[16] T. Stockhammer, M.M. Hannuksela, and T. Wiegand. H.264/AVC in wireless environments. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):657–673, July 2003.

[17] 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.