An H.264/SVC-based adaptation proxy on a WiFi router

Ingo Kofler, Martin Prangl, Robert Kuschnig, Hermann Hellwagner
Institute of Information Technology
Klagenfurt University
Universitätsstraße 65–67, Klagenfurt, Austria
{firstname.lastname}@itec.uni-klu.ac.at

ABSTRACT

Recent advances in video coding technology like the scalable extension of the MPEG-4 AVC/H.264 video coding standard pave the way for computationally cheap adaptation of video content. In this paper we present our work on a lightweight RTSP/RTP proxy that enables in-network stream processing. Based on an off-the-shelf wireless router that runs a Linux-based firmware we demonstrate that the video adaptation can be performed on-the-fly directly on a network device. The paper covers design and implementation details of the proxy as well as a discussion about the actual adaptation of the SVC stream. Based on experimental evaluations we show that our approach can handle a reasonable number of concurrent sessions for a typical home deployment scenario. Furthermore, the paper covers possible applications in which adaptation on the network device can be beneficial.

Categories and Subject Descriptors

C.2.6 [**Computer Communication Networks**]: Internetworking; H.4.3 [**Information Systems Applications**]: Communications Applications; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems

Keywords

Multimedia adaptation, in-network adaptation, RTSP, RTP, H.264, scalable video coding

1. INTRODUCTION

In-network adaptation of multimedia streams may be beneficial in situations when several clients with diverse characteristics have to be served or when a gateway has to convey the streams to a downstream network offering lower bandwidth or suffering from congestion or increased packet losses; often, wireless access networks are subject to such conditions. Scalable video coding, more precisely the scalable extension of the MPEG-4 AVC/H.264 video coding standard (H.264/SVC) [1], now offers the option to perform computationally cheap video stream adaptation. The layered video encoding allows to adapt the video by truncating certain parts of the bitstream. As a consequence, the interesting research question arises, whether or not video adaptation based on H.264/SVC becomes feasible on low-end network devices like an off-the-shelf wireless router.

In this work, we perform an investigation into, and demonstrate, the feasibility of H.264/SVC stream adaptation on a WiFi router running a lightweight Linux distribution. The SVC-codec specific adaptation is performed on-the-fly by an application-level proxy process on the WiFi router that intercepts the RTSP/RTP video communication between a video server and clients requesting streams. The design considerations and decisions that led to our prototype proxy are discussed; these are mainly motivated by the attempt to make the proxy *transparent* for the regular RTSP/RTP server–client communication. The usage of a proxy offers the possibility to be session-aware and allows to differentiate and handle incoming RTP streams (audio, scalable video) appropriately. The performance results indicate that, despite the user-space proxy implementation and poor hardware capabilities, adaptation of a few parallel H.264/SVC video streams and possible deployment in a home multimedia scenario is well feasible.

The paper is organized as follows. Section 2 introduces the network protocols related to RTSP/RTP-based media streaming, in particular some aspects that are important in the context of this paper. Our approach for adapting SVC-based video streams, which goes beyond the capabilities of a simple packet dropping scheme, and the actual design of the proxy are given in Section 3. The hardware and software platforms that were used to develop and deploy our adaptation proxy are described in Section 4. Using this implementation, we conducted a performance evaluation, the results of which are given in Section 5. The fact that video streams can be adapted directly on the network device enables a variety of applications that are briefly discussed in Section 6. Section 7 concludes the paper.

2. PROTOCOL BASICS

2.1 RTSP

The Real Time Streaming Protocol (RTSP) (RFC 2326) was developed for interactions between a media server and a consuming client, with a focus on session control (VCR-like) commands from the client, e.g., play, pause, etc. RTSP messages can be transported over UDP or (mostly) TCP.

NOSSDAV '08 Braunschweig, Germany

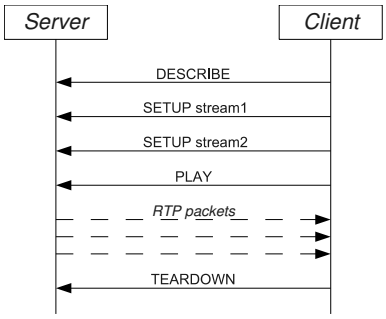


Figure 1: Typical RTSP interaction

The server is listening for the client's RTSP requests (commands). The message passing between the client and the server is similar to HTTP. The main difference is that the server tracks state information for each RTSP session. The media data itself is typically transported over RTP/UDP which is used in tight combination with RTSP. A typical RTSP/RTP client–server interaction is shown in Figure 1. It contains the important RTSP commands which are relevant to this work, briefly discussed in the following.

The DESCRIBE command is used for describing the media presentation which is identified by a unique RTSP URL. The reply includes the presentation description which is typically encapsulated in the Session Description Protocol (SDP) (RFC 4566) format. It includes a list of media streams of the selected presentation. In most cases, there is one media stream for audio and one for video. The SETUP request specifies how a media stream has to be delivered from the server to the client. It includes the required media transport protocol (in our case RTP) as well as the two reserved data ports at client side. One port is used for the receipt of the RTP stream, the other one for RTCP feedback. The server replies and confirms the chosen parameters and fills in additional information for the client, e.g., the server's chosen port for the client's feedback (RTCP). The SETUP request has to be submitted for each elementary stream separately before an aggregate PLAY command can be sent. A PLAY request invokes the media server to deliver a media stream to the client. The media stream is segmented and encapsulated into RTP packets which are subsequently sent to the negotiated client's data port. A TEARDOWN request invokes the termination of the session. It stops all media streams and frees all session related data (state information) on the server.

2.2 RTP w.r.t. H.264/SVC

The Real-Time Transport Protocol (RTP) (RFC 3550) enables end-to-end transmission of real-time data, such as audio or video, over multicast or unicast networks. Its design is independent of the used transport layer. Data is transmitted in a packet-oriented way. Feedback from the RTP clients via the RTP Control Protocol (RTCP) can be used to monitor the data delivery. Each RTP packet consists of an RTP header and the payload data. The main parameters of the RTP header are the *Payload Type*, *Sequence Number* (used for in-order delivery), *Marker Bit* (signals payload specific events like the last packet of a video frame), *Timestamp* (display timestamp of the data), and *Synchronization Source (SSRC)* (for media source identification). The way the media stream is encapsulated in the RTP payload is specific for each payload type.

H.264/AVC video uses the Network Abstraction Layer (NAL) [1] to handle any content in a common fashion. Each part of the media stream is encapsulated into NAL units (NALUs). The NAL units are distinguished from each other by means of the NAL unit type, which is included in the 1-byte header preceding each NALU. H.264/SVC, the scalable extension of H.264/AVC, additionally offers scalability in the temporal, spatial, and quality domains. The SVC NALU header, as shown in Listing 1, extends the AVC NALU header with a 3-byte extension header.

Listing 1: NALU header with SVC extension

```
+-------- .. -+------ .. -+---------------+---------------+
|0|1|2|3  ..  |7|0|1|2  ..  |7|0|1|2|3|4|5|6|7|0|1|2|3|4|5|6|7|
+-+-+-+-+ .. -+-+-+-+ .. -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|F|NRI| Type |R|I| PRID |N| DID |   QID   | TID |U|D|O| RR|
+-------- .. -+------ .. -+---------------+---------------+
```

H.264/SVC offers scalability on bitstream level, meaning that the quality and bitrate of the video can be adjusted by simply discarding NALUs of a specified layer. The most important fields w.r.t. scalability are the temporal id (TID), dependency id (DID), and quality id (QID). SVC NAL units having the same (TID, DID, QID) triple form an enhancement layer. A detailed description of the NALU header parameters can be found in [1]. The base layer of a scalable bitstream consists of AVC NALUs, which are preceded by special Prefix NALUs to provide scalability information.

RTP packetization of H.264/AVC content [2] is based on the NALU concept. Depending on the size of the RTP packet and NALU, one or more NALUs can be packed/aggregated into an RTP packet payload. Single time aggregation packets (STAPs) allow NALUs with the same timestamp to be packed together. The payload of STAP-A (NALU type 24) consists of a 1-byte NALU header followed by the size of the first NALU (2 bytes) and the actual NALU. Subsequently, the size of the next NALU and the NALU itself is attached to the RTP payload, and so on. This enables fast access to a specific NALU header in the RTP packet and so simplifies processing. If the size of a NALU exceeds the RTP payload size, it has to be fragmented into fragmentation units (FUs) and consecutively packed into RTP packets. The FU-A fragmentation mode (NALU type 28) consists of a 1-byte NALU header (FU indicator) followed by the 1-byte FU header. The FU indicator is used to identify the RTP payload as a fragmentation unit and the FU header signals wether the fragment is the first or the last. Additionally, the FU header supplies the NALU type of the fragmented NAL unit. For legacy reasons, the indication of the packetization mode is realized by a 1-byte NALU header in the RTP payload. The NALU types were selected in a way that they do not interfere with those defined in the base H.264/AVC specification. The STAP-A and FU-A packetization mode is illustrated in Figure 2. More aggregation modes like MTAP (multi-time aggregation) or enhanced techniques like interleaving are described in [2].

Just as the H.264/SVC codec extends H.264/AVC, RTP packetization of H.264/SVC content also extends H.264/AVC packetization [3]. The aggregation and fragmentation modes work in the same manner and sadly regard the SVC extension header as payload data. Problems arise with the use of fragmentation units, where only the first fragment includes the SVC header. Additionally the new Payload

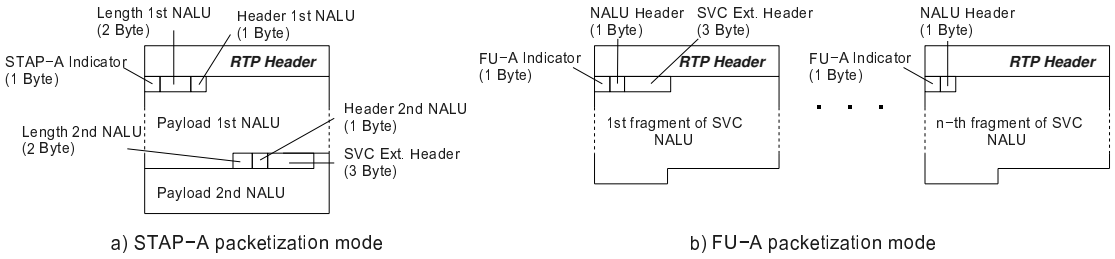


Figure 2: Typical H.264/SVC packetization modes

Content Scalability Information (PACSI) NALU was introduced, which serves as a table of contents for an aggregation packet.

2.3 Deployment issues

In the past media transport using RTP led to problems in cases where firewalls and home gateway network nodes were deployed. The reason lies in the nature of RTP, that is often used on top of the connectionless UDP protocol, in combination with the fact that the media server starts the RTP data traffic to the client by sending UDP datagrams to a negotiated (via RTSP SETUP) port at the client side. In order to handle the RTP streams appropriately at firewalls and devices that perform Network Address Translation (NAT), it is necessary that these devices perform a stateful inspection of the RTSP traffic. If this is not the case, the network device is not able to associate the incoming RTP streams to an established RTSP session and will drop the incoming packets.

These drawbacks are well known in the community and several approaches for tackling the NAT traversal problem are known. Currently, there exists a draft of the so called NSIS Signaling Layer Protocol (NSLP) [4], which is designed to request the dynamic configuration of firewalls and NAT routers along the data path. Another effort of the IETF is the standardization of the STUN protocol (RFC 3489) that allows for detecting NAT devices and the traversal of them. STUN is also considered in the upcoming RTSP 2.0 protocol where it is used in combination with TURN and ICE for dealing with NAT devices. However, except STUN all other protocols are currently still IETF drafts. Another approach is followed in the scope of the Universal Plug-and-Play (UPnP) framework [5]. There, clients like a media player can make use of services offered by routers or NAT devices and can configure them to pass through RTP streams to the player. However, such automatic configuration mechanisms are often criticized for security reasons.

3. DESIGN OF THE ADAPTATION PROXY

3.1 Architecture

The adaptation proxy that we developed for H.264/SVC video adaptation follows a very modular design. The main building blocks of the proxy and its corresponding interfaces are depicted in Figure 3.

The *RTSP Handler* is responsible for serving RTSP requests from the client and for modifying and forwarding them to the original media server. Although RTSP can be used both over UDP and TCP connections, we currently focus only on a TCP implementation. The RTSP handler was implemented in a multi-threaded fashion as it creates a separate thread for handling each incoming TCP connection. Although we tried to reduce the number of threads and context switches, we took this decision to have a simpler implementation for handling the concurrent, persistent connections.

The *SDP Cache* is used for caching SDP descriptions that are returned in response to RTSP DESCRIBE requests. DESCRIBE requests are state-less and do not lead to session establishment. This means that a client could issue DESCRIBE requests for different resources on the server without sending a subsequent SETUP request. So the proxy cannot associate an SDP description with a certain session since at that moment no session exists. However, the proxy cannot simply discard an SDP description since the information contained in it (e.g., contained media streams) is required later on. The proxy therefore caches the SDP descriptions in an LRU-cache which holds a configurable number of descriptions. To enable a unique identification of each SDP description, the URI of the corresponding resource is used.

A central component of the proxy is the *Session Manager*. Its task is to maintain the sessions and all corresponding media tracks. Further, it holds references to all session-related data structures like socket connections, statistics, and timing information. It also keeps track of the activity of the sessions and terminates sessions that are inactive for a certain amount of time.

The actual handling of the RTP streams that are carrying the media data is done within the *RTP Forwarder/SVC Adapter*. In this context, the term forward does not refer to forwarding of packets at the IP layer but to receive incoming RTP packets from the server and to send them towards the client. In contrast to IP forwarding, this is performed on the application layer within the proxy. Ordinary non-SVC based streams are handled by this component by receiving a packet from a UDP socket, modifying only parts of the RTP header (sequence numbers, timestamps, SSRC, etc.) and sending it to the client. Additionally, SVC-based streams can be adapted by the SVC Adapter on a per-packet basis. This is done by utilizing the SVC-specific RTP packetization and is further described in Section 3.3. In contrast to RTSP session handling, this component is single-threaded in order to avoid context switches when handling packets of different sessions. Concurrency between the sessions is achieved by using non-blocking I/O for receiving incoming packets. In close co-operation with the RTP Forwarder, the *RTCP Handler* controls both the receipt and transmission of RTCP reports. RTCP handling is not as easy as just

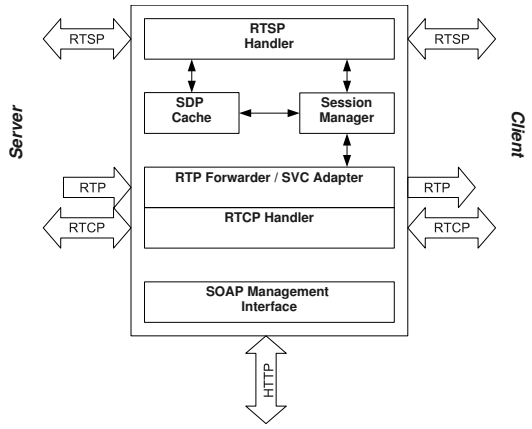


Figure 3: Adaptation proxy architecture

forwarding reports from the client to the server and vice versa. Since the proxy is both client and server for two distinct RTSP sessions, it has to consider different timestamps and SSRCs in both sessions. Furthermore, the adaptation of SVC content at the proxy makes it necessary to provide correct sender reports which reflect the reduced number of bytes and packets sent to the client.

The *SOAP Management Interface* offers an API to the proxy which can be used for both retrieving status information about the proxy and its current sessions and for configuring adaptation parameters (TID, DID, QID) for the SVC streams. At the current state of implementation this interface is used for monitoring and debugging the proxy's operation. In our future work this interface will be used by a control application that runs also on the wireless router and determines the optimum adaptation parameters.

3.2 Proxy interaction

The participation of the proxy in the case of media streaming is as follows. On the wireless router, a dedicated firewall rule is used to redirect each RTSP connection to the proxy instead of forwarding it to the media server. The handling of the incoming connection is performed by the RTSP handler which behaves like a server for the client. The proxy itself connects to the media server and acts as a client. As a result, two distinct RTSP sessions exist: one between the server and the proxy, and another one between the proxy and the client. After the establishment of the session it is up to the proxy to process the client's requests, to modify them appropriately and to forward the modified requests to the server. The necessity of modifying the request arises since parts of the request are related to the RTSP session between proxy and client (e.g., the session identifier). Also the UDP ports for RTP/RTCP are negotiated separately in each RTSP session which makes it necessary to modify the RTSP SETUP request. As the proxy modifies the request before sending it to the original server, it can be classified as intercepting proxy.

3.3 SVC adaptation

The adaptation of the scalable video stream is performed directly on the RTP stream on a per-packet basis. This implies that the proxy does not depacketize the NALUs and aggregate them to a complete access unit for the task of adaptation. This reduces the amount of memory required by the proxy, which is typically a scarce resource on small devices. Besides, memory copy operations within the user-space application are avoided by operating only on a single buffer that is used for receiving packets via the UDP socket API, performing the adaptation, and sending the (possibly adapted) packet via the socket API to the client.

The adaptation is steered by specifying adaptation parameters in terms of a (TID, DID, QID) combination. Based on these adaptation parameters, all NALUs that are not satisfying the adaptation parameters are removed from the RTP stream. In our approach, this is done by either discarding the complete packet or by removing certain parts of the packet. Although the adaptation is done on a per-packet basis, it is not based on a state-less inspection of each packet header. The SVC payload format (examples in Figure 2) has two characteristics which make it difficult to adapt or drop packets without maintaining state.

First, the layer information of each SVC NALU is signaled in a 3-byte header extension of the 1-byte NALU header. However, in the case of fragmentation only the 1-byte NALU header is repeated in each RTP packet. This implies that a stateless packet inspection is not able to determine the corresponding layer information for a given FU-A packet.

Second, for backward compatibility, NALUs which belong to the AVC base layer do not contain any scalability information but use prefix NALUs to signal which temporal layer they belong to. For packetization it is recommended that a prefix NALU should be transmitted in the same RTP packet as its associated AVC NALU. However, we discovered that this is not always possible as AVC NALUs tend to become significantly larger than the MTU and are therefore packetized using fragmentation units (FUs). In that case, the prefix NALU cannot be transmitted in the same RTP packet and requires a stateful packet inspection as well.

The adaptation of the SVC stream works as follows. The proxy receives an RTP packet via a buffer that is provided to the UDP socket library. After successful receipt of the packet, the RTP header and the payload structure are analyzed. In the case of fragmented NALUs, the adaptation of the stream is quite simple. Based on the extension header that is contained in the first fragment, the proxy decides wheter the NALU should be forwarded to the client or discarded. If the decision is that the NALU has to be discarded, an internal flag is set which reminds the proxy that all other fragments belonging to that specific NALU have to be dropped as well. In the case that the NALU should not be discarded by the proxy, the proxy modifies the RTP header appropriately (sequence number etc.) and sends the data contained in the buffer to the client.

The adaptation of RTP packets that follow the STAP-A packetization mode is done as follows. The proxy starts at the first NALU in the RTP packet and determines its length and layer information. If the NALU should be included in the adapted bitstream, the proxy jumps to the next NALU by utilizing the length information. This inspection of the NALU header is done iteratively as long as there are NALUs left in the packet and no NALU is discovered that does not match the (TID, DID, QID) adaptation parameters. If none of the NALUs contained in the RTP packet has to be removed, the proxy updates only the RTP header of the packet and sends the buffer to the client. If some of the NALUs at the end of the packet have to be fil-

tered out, the proxy removes them by simply not sending the complete received buffer, but only the relevant first N bytes to the client. However, if it turns out that not even a single NALU of the packet should be delivered to the client, the whole packet is dropped.

In addition to the mechanisms described above, the proxy is also aware of prefix NALUs. If a prefix NALU is encountered, the layer information is extracted and used for handling the subsequent AVC NALU. Since this subsequent NALU can be either in the same packet or not, the proxy uses some internal flags for keeping this state information.

3.4 Assumptions and limitations

In our current work, we focus on adapting streams that are packetized using both the fragmentation unit (FU-A) mode and the non-interleaved single time aggregation (STAP-A) mode. The adaptation of packets that are using multi-time aggregation (MTAP) was intentionally not considered since aggregation of NALUs belonging to different timestamps does only make sense for applications with very low bitrates. The adaptation of AVC/SVC streams that use the interleaved packetization modes (STAP-B, FU-B) and PACSI NALUs [3] are currently not supported but will be tackled in the future.

4. IMPLEMENTATION PLATFORM

The platform that was selected for deploying and evaluating our lightweight RTSP/RTP proxy is a Linksys WRT54GL Wireless-G Broadband Router. The reason for selecting this device is that the original firmware can be easily replaced by a variety of different third-party firmwares. One of the most popular firmwares is OpenWRT[1], a small-scale Linux distribution tailored for embedded devices. Although the name suggests that it is dedicated for the Linksys WRT routers, it can be used on a variety of different wireless routers. It features a fully writable filesystem, an ordinary Unix-style shell, and even a package management system.

OpenWRT offers a dedicated SDK for developing applications or libraries. The SDK consists of a cross-compiler tool chain which can be used to compile existing C or C++ applications or libraries for OpenWRT. At the end of the cross-compilation a package is created which can be either installed via the package management software or directly integrated in the firmware image. The implementation of our proxy is based on plain ANSI C and is kept as lean as possible. Also the usage of third-party libraries was kept at a minimum. The proxy only makes use of the standard socket API, the POSIX thread (pthread) library, and the gSOAP[2] library.

From a hardware point of view the Linksys WRT54GL Wireless-G Broadband Router is based on the Broadcom SoC (system-on-chip) BCM5352EL. It consists of a MIPS32 processor running at 200 MHz, an IEEE 802.11b/g MAC/-PHY, an SDRAM controller, and a configurable Fast Ethernet switch with five ports. The router offers in total 16 Mbytes of main memory from which the majority is used by the kernel, necessary daemons (SSH server, HTTP server), and a writable, temporary filesystem (tempfs). The amount of memory available for user-space processes depends on the actual configuration (kernel options, running daemons).

[1]http://www.openwrt.org
[2]http://gsoap2.sourceforge.net

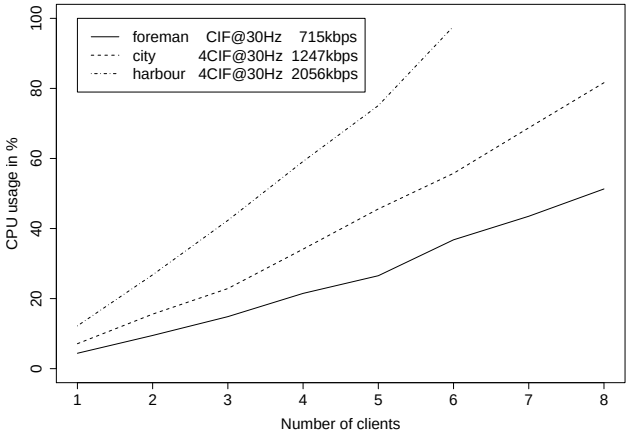


Figure 4: CPU usage of adaptation proxy process

In our configuration about 6 Mbytes were reported as free memory. The WRT54GL is equipped with a 4 Mbytes flash memory which contains the OpenWRT distribution. Due to the limited storage space, major parts of the distribution are located in a compressed read-only filesystem (squash-fs).

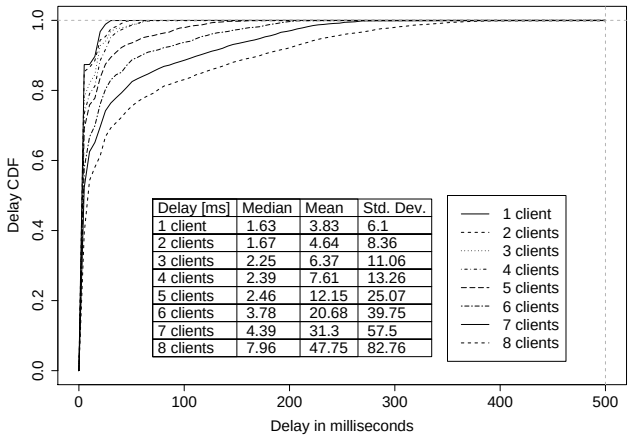
5. EVALUATION RESULTS

Based on the hardware and software platform introduced above, we performed an experimental evaluation for figuring out which limitations the low-end hardware imposes on the real-time adaptation of H.264/SVC video streams. As a scalability metric, the CPU usage of the proxy process on the wireless router was used when forwarding and adapting an increasing number of streams. The distribution of the end-to-end transmission delays of video frames was considered as a metric relevant for the user experience. An SVC-enabled version of the Darwin Streaming Server[3] was used as an RTSP/RTP server and Live555[4] as an RTSP/RTP client. For the measurements three representative video sequences (*foreman*, *city*, and *harbour*) were selected, which were encoded with up to 8 layers at average bitrates of 715, 1247 and 2056 kbps, respectively. Each client was receiving the same content from the streaming server via the proxy.

The delay between the streaming server and the client is measured on a picture-by-picture basis. So after fully sending/receiving the last NALU of a picture, the timestamp for the picture is recorded. This is done for each stream served by the streaming server and on each client. The test sequences were being played in a loop for a total of 54000 pictures, which results in a streaming and playout time of 30 minutes. The adaptation parameters for the adaptation process were selected in such a manner that all packets are passed through. This would be the worst case scenario because all of the NALUs have to be inspected by the proxy and transmitted to the client.

The CPU usage in Figure 4 shows that with this prototype implementation it is possible to adapt up to five *harbour* streams with a cumulated bandwidth for all clients of 10 Mbps. For the test sequence *city* we can safely assume

[3]http://developer.apple.com/opensource/server/streaming
[4]http://www.live555.com/liveMedia



Delay [ms]	Median	Mean	Std. Dev.
1 client	1.63	3.83	6.1
2 clients	1.67	4.64	8.36
3 clients	2.25	6.37	11.06
4 clients	2.39	7.61	13.26
5 clients	2.46	12.15	25.07
6 clients	3.78	20.68	39.75
7 clients	4.39	31.3	57.5
8 clients	7.96	47.75	82.76

Figure 5: Delay distribution (CDF) for test sequence *city*

that the processing of four streams by the router will not restrict its routing functionality because the CPU usage (35%) is quite moderate. It can be observed that the CPU usage of the adaptation proxy scales almost linearly with the number of streams. The maximum transmission delay (see Figure 5) for *city* with four clients is below 100 ms. With higher load (more than four clients) the transmission delays rapidly increase and so does the transmission delay jitter. While showing the processing limitations due to application layer processing of the RTP packets, this evaluation can point out that the induced transmission delay for *city* (1247 kbps) and four clients is clearly smaller than 3 frame times.

6. APPLICATIONS

As we demonstrated in the previous section, depending on the actual bitrate of the video our RTSP/RTP proxy is able to handle a reasonable number of streams in parallel. The possibility of deploying such a proxy on existing, cheap network devices offers a variety of application scenarios for typical home deployments.

An important application of performing SVC adaptation on the wireless router or access point (AP) is cross-layer adaptation. Since the AP is aware of all wireless stations that are associated with it, it can determine which video streams are transmitted over the wireless interface. In case of bad wireless conditions between the AP and the receiving station, the video can be adapted to meet the actual limitations of the link, e.g., the actual physical rate. The adaptation parameters can be provided by a separate process on the wireless router that actually monitors the conditions and controls the adaptation via the SOAP management interface. Our future work will focus on this scenario.

A further interesting application could be the automatic adaptation of audio-visual streams for different devices within the home network. A consumer of multimedia content might have a variety of devices (smart phone, desktop PC, etc.) with different playback characteristics (e.g., bit rate constraints, display resolution). These characteristics can be described by an MPEG-21 Usage Environment Description (UED) and stored at the wireless router as a kind of device profile. The association between the device and its profile can be realized by the device's MAC address, since in a home deployment all devices are typically directly connected to the AP by either an Ethernet or 802.11 link. Based on the information from which device an RTSP request was issued and the device profile stored at the wireless router, an automatic adaptation of the video content according to the device capabilities can be accomplished.

Finally, the deployment of such a proxy solves the NAT traversal problem with RTSP/RTP-based streaming. In common home deployments the wireless router is typically the network entity that holds the public IP address of a broadband connection and connects the private home network to the Internet by performing NAT. The deployment of a proxy on the NAT device solves the traversal problem since the UDP ports are negotiated separately between the client and the proxy, and the proxy and the server, and the ports offered by the proxy are available at its public IP address. In contrast to other techniques the proxy approach does not require any modification on the client or the server.

7. CONCLUSIONS

In this paper we presented our work and achievements on a lightweight RTSP/RTP proxy implementation that can be deployed on an off-the-shelf wireless router. The proxy is able to adapt H.264/SVC streams on a per-packet basis which enables a higher flexibility compared to a simple priority-based packet dropping mechanism. In experimental evaluations we were able to show that the processing capabilities of the router are sufficient for performing in-network adaptation of four simultaneous video streams with a CPU usage of only 35 percent while introducing only a low delay on the video stream. As a by-product, the proxy approach provides a solution for the RTSP/RTP NAT traversal problem for typical home network deployments. Although our current work focuses mainly on the actual adaptation mechanism, the paper points out possible applications in which video adaptation on such an embedded device is beneficial.

8. ACKNOWLEDGMENTS

This work was supported by the EC in the context of the ENTHRONE II project (IST-1-507637; http://www.ist-enthrone.org) and by the Austrian Science Fund (FWF) under project "Adaptive Streaming of Secure Scalable Wavelet-based Video (P19159)".

9. REFERENCES

[1] T. Wiegand, J. Ohm, G. Sullivan, and A. Luthra. Special Issue on Scalable Video Coding – Standardization and Beyond. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9), September 2007.

[2] S. Wenger et al. RTP Payload Format for H.264 Video, RFC 3984. February 2005.

[3] S. Wenger et al. RTP Payload Format for SVC Video, Internet Draft. January 2008.

[4] NAT/Firewall NSIS Signaling Layer Protocol (NSLP). 2007. NSIS Working Group Internet Draft, version 14.

[5] UPnP Forum. Internet Gateway Device (IGD) Standardized Device Control Protocol. November 2001. version 1.0.