GENERIC QOS AWARE MEDIA STREAM TRANSCODING AND ADAPTATION

Andreas Kassler, Andreas Schorr

Department of Distributed Systems, University of Ulm, Germany.
E-mail:kassler@informatik.uni-ulm.de|schorr@informatik.uni-ulm.de.

ABSTRACT

In this paper we propose a generic and highly efficient QoS aware media adaptation and transcoding service. Based on a flexible component model, we have built adaptable network based media adaptation units that allow to change the codec and the quality of compressed media streams on demand during their transmission. Using a combination of sender rate adaptation, filtering and transcoding inside the transmission path, and receiver adaptation we can achieve an appropriate tradeoff between flexibility, bandwidth efficiency and security. Several media adaptation units together form an application layer overlay adaptation and transcoding network to support heterogeneous multimedia communication. We provide several measurements to evaluate the scalability of our approach both with respect to processing requirement and bandwidth scalability. We show that when using proper application level framing, quality adaptation can be achieved in real-time for several clients in parallel.

1. INTRODUCTION

The heterogeneity of current and future networks and operating system environments is increasing. End-systems and networks show a great variety of characteristics and levels of performance. Local and network resource availability will be limited and applications may suffer from these variations. Mobility of terminals and users will increase the heterogeneity and make it more difficult to provide predictable end-to-end Quality of Service (QoS) as the user perceives it. Therefore, applications should be able to adapt themselves and adjust their resource demands and capabilities dynamically to cope with changes in environmental conditions.

Typically, distributed multimedia applications use feedback information to adapt, but cannot provide fairness to all applications as global information is not available. On the other hand, resource management systems built into operating systems do not have the knowledge of the semantics of the applications that use the resources. However, before even establishing media sessions, a common set of capabilities has to be negotiated. If no common ground can be reached, communication is only possible using intermediate transcoders, that convert the media representation of the sender to a format the receiver can deal with. Transcoding may be required at several points in the transmission chain e.g. converting from a camera format to an editing format or converting from a high bit rate production format to a low bit rate distribution format. Transcoding differs from the encoding process because a transcoder only has access to an already compressed signal by reading the output of an encoder or a previous transcoder.

We believe that media adaptation and transcoding is a key technology to bridge the heterogeneity gap. Nevertheless, an interaction of media, mobility and resource management is necessary to provide predictable end-to-end QoS, which is hard to achieve due to increased heterogeneity. One example of such an integrated framework was developed within the MASA project (Mobility And Service Adaptation) [1], [2]. In this paper, we describe the architecture of a media adaptation unit which allows to adapt the quality of compressed media streams and change the codec on demand. Our solution is based on a reusable component based architecture, by which we are able to build adaptive and QoS aware media adaptation units. The MAUs form an application layer overlay transcoding and adaptation network to support a wide variety of heterogeneous end-systems.

The rest of the paper is organized as follows: Chapter 2 provides an overview on potential conflicts due to heterogeneity and motivates our work. Chapter 3 provides background information on mechanisms used and related work. The architecture of the media adaptation unit is introduced in chapter 4. Chapter 5 provides several measurements to evaluate our approach with respect to processing requirements and scalability. Finally, a summary in chapter 6 concludes this paper.

2. HETEROGENEITY IN DISTRIBUTED MULTIMEDIA SYSTEMS

Distributed multimedia systems are inherently heterogeneous. The networks provide support ranging from a few kbps up to several mbps even on the air interface, packet loss rates range from almost loss less transmission for the fixed network to error prone wireless environment. End-systems have different processing capabilities that range from low power PDAs to high performance workstations. Software (e.g. codec availability) and hardware (e.g. display sizes) capabilities of the end-systems are highly heterogeneous. The heterogeneity is not only static but also *dynamic* [2], as software capabilities, resource availability and resource requirements may change over time. For example, software capabilities can be enhanced by dynamically downloading software codecs. Also, user requirements may change even during communication sessions. In a multi-homed terminal, the network availability may change as the user moves around. *Always best connected* at the best quality is one important user goal. For instance, low power mobile devices and high power workstations may participate in video conferencing scenarios, simultaneously. Mobile devices may perform hand-over procedures during on-going multimedia sessions to change the access networks. Bridging this heterogeneity gap is a major issue in multicast communication scenarios [3].

There has been a substantial amount of research on methodologies for efficient use of multicast in VoD scenarios. Peroidic broadcast and patching [4] are two approaches which use multiple multicast channels to reduce the network and server resource consumption, while at the same time satisfying the asynchronous requests of individual clients. Internet VoD Cache Servers were proposed [5] to further enhance the availability of VoD services. Nevertheless these approaches do not address the heterogeneity problem that arises when several receivers of the same movie expect different quality parameters or codecs of the video streams. Also, these approaches require considerable amounts of disk space and high bandwidth network links on the client terminals. Our approach presented in this paper addresses any kinds of clients even small hand-held devices with limited resources, and with different capabilities. Furthermore we consider not only VoD services, but also live video streams.

Different solutions address the problem to bridge static and dynamic heterogeneity. In a heterogeneous conferencing scenario, an agreement for the capabilities that all the peers can use is necessary to establish communication. If some clients can only decode MPEG-4 [6] streams whereas other peers can only understand the WaveVideo (a Wavelet based video codec) [7], a communication is either not possible or a transcoding module is necessary that changes the media codec to a format the receiver can cope with. A number of ways to address the problem are described in [8], [9], [10], [11], and [12].

The filtering approach ([11], [12]) seems to be most promising as the advantages of multicast delivery are combined with quality selection. However, the number of quality levels obtained by filtering MPEG streams is limited and significant processing time is required. If in addition the receivers are interested in different QoS adaptation policies in case the negotiated QoS cannot be maintained, the problem becomes more complicated. For example, one receiver could be interested in maintaining the frame rate while degrading the frame quality whereas another receiver would like to preserve the frame quality. Finally, in the filtering approach, the filters must also be able to change the format of the stream. We use and extend the filtering approach as a basis for our system as it is the most flexible solution.

3. MEDIA STREAM ADAPTATION MECHANISMS

Adaptivity is beneficial as it provides high flexibility. Adapting media streams influences user perceived quality and resource requirements both for the endsystems and for the network. Sender rate adaptation offers the highest flexibility, since the full semantical information of the data source is available and the sender has full control over its data. Receiver adaptation is useful in multicast scenarios to reduce processing requirements at the receiver, if he has not enough resources to handle the original stream. Network based media adaptation can adapt the stream to the aggregated demands of the following subtree, which bridges the heterogeneity gap for multicast scenarios (the filtering approach). Mid-term, slow adaptation to the static demands of the session can be achieved by using appropriate feedback mechanisms in a distributed set of media adaptation nodes (filter propagation, [12]). To deal with highly dynamic changes of resource availability, packet filtering mechanisms inside routers can be used to allow for short-term, very fast, locally optimized reactions and seamless handoff performance on mobile clients (e.g. in case of queue buffer overflows). Finally, user perceived QoS needs to be managed and maintained end-to-end. This requires local, peer and network resource reservation, adaptive media management to react properly to QoS violations and a coordination of the activities to comply to users requirements. As it is too complex to manage these tasks by an application itself, a QoS framework (like the MASA QoS framework [1]) is necessary that applications can use to provide QoS enhanced adaptive multimedia services towards the user.

There are different possibilities to realize scaled media streaming. Receiver driven layered multicast (RLM) [13] is a promising approach. Receivers can decide about the quality by simply joining or leaving respective quality levels. The original stream is decomposed into a hierarchy of sub-streams, each adding information. Each sub-stream is sent to a different multicast group, so receivers essentially join or leave a set of multicast groups. However, without inserting redundant layers, such fixed layer subscription schemes are not able to support different receiver QoS adaptation policies at the same time.

For sender rate adaptation, adaptive codecs have been proposed (e.g. layered DCT [14], *lH.261* [15]). While these approaches are useful in a point-to-point scenario, they often suffer from the flexibility for the support of different receiver requirements. Wavelet based coding mechanisms seem to be very promising due to their inherent scalability. In this paper we are using the wavelet based WaveVideo codec developed at the ETH Zürich [7], which offers a good compromise between compression ratio, error tolerance, implementation efficiency, and the ability to support various filter operations. By using a layered hierarchical coding scheme and proper application layer framing (ALF) [16], WaveVideo is able to support prioritization of packets and fast adaptation. Nevertheless, one problem remains (also for the RLM approach): if no agreement on a common set of codecs can be reached, communication is still not possible.

We have proposed a set of mechanisms [17], [18] to adapt the quality of compressed WaveVideo streams in real time based on packet dropping strategies. The main idea is that each network packet contains a 32-bit tag that describes the content of the packet. A filter module inspects the tag and decides to drop or forward the packet to the downstream node. We developed several semantical filters at the application layer to adapt the frame rate, frame size, visual quality. In [19] we have proposed a dynamic rate shaping filter that takes into account user preferences and adapts itself to maintain a given target rate. It is however required, that the encoder creates the correct tags, inserts them after the RTP [20] header and separates the packets according to quality dimensions. For example, packets contributing to the luminance quality would be assigned a different tag than packets contributing to the chrominance quality. This paper integrates the adaptation mechanism with transcoding.

The most simple approach to transcoding is to decode the input bit stream and then recoding with a standalone coder to the desired output format and bit rate. However, each additional transcoding step introduces noise. For an overview on optimizing a single transcoding step, see for example [21]. In addition, such optimization mechanisms are typically restricted to a certain family of codecs, e.g. codecs that use the DCT. As there are many possible combinations of input/output codecs that might each be optimized, we are more concerned about building an architecture for generic transcoding. Nevertheless, we can incorporate such optimizations in our architecture.

4. ARCHITECTURE

We propose a generic media adaptation unit which is able to transcode media streams from any format into a format the receiver can understand and adapt the stream to fit the clients resource availability and QoS requirements. This is motivated by the following reasons:

- Adaptation and transcoding may be required to bridge the heterogeneity gap.

- It is important to minimize additional overhead for servers and clients. For scalability reasons, VoD servers should not be concerned about additional adaptation/transcoding. Low power PDAs often do not have hardware support to decode the original stream. Therefore, on demand transcoding and adaptation of the quality should be performed on intermediate media adaptation units.

- Adaptation *on the fly* and transcoding is feasible on today's hardware. It is more efficient wrt. to data storage requirements to compute a different representation on demand rather than storing a set of pre-adapted representations and switching between them during transmission.

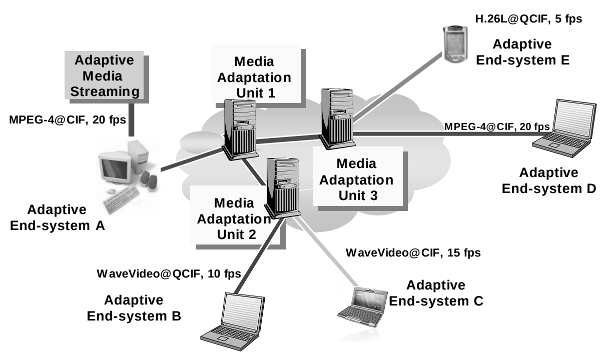

Fig. 1. Overall Architecture: Media Adaptation Units as an overlay distribution network.

Our overall architecture is depicted in Figure 1. Adaptive media clients communicate with each other using application layer transcoding and media stream adaptation services provided by media adaptation units (MAUs) distributed across the network. Thus the MAUs form an application layer overlay content distribution and adaptation network. In this figure, 3 MAUs transcode and adapt the stream emitted by adaptive end-system A. We propose to use the MASA QoS framework on each adaptive endsystem. This enables an integration of local resource management, network resource management, mobility management with media adaptation at the endsystem. As each MASA node monitors its resources and manages the media stream compression and transmission, we focus in this paper on the Media Adaptation Units. For more information on MASA, see [1], [2].

4.1. Media Adaptation Unit Architecture

When designing the architecture of the media adaptation unit we had several issues in mind.

- *Component Re-usability*: to minimize implementation effort by re-using components.

- *Generic Architecture*: to support media stream transcoding as well as media adaptation.

- *Flexible and extendable*: to support future codecs and optimized transcoders, that may be dynamically downloaded to extend functionality on the fly.

- *QoS awareness*: to be aware of QoS constraints and enforce them.

- *Multi-stream capability*: to support many transcoding/adaptation processes in parallel.

- *Rate and Error Control*: to control the rate of the compressed streams in order to implement e.g. TCP-friendly rate control. Also, error control is an issue in wireless networks and MAUs should be able to retransmit lost packets.

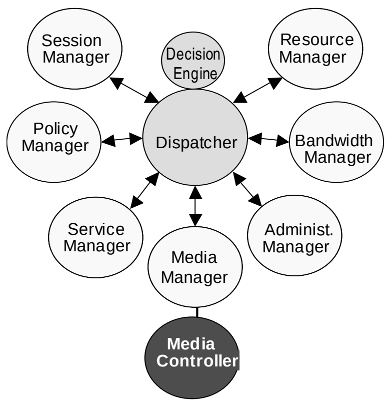

Fig. 2. Architecture of the media adaptation unit.

We use a manager/controller design pattern [22] to describe the architecture of the MAU (see Figure 2), where managers provide a message based interface for a subsystem responsible for a dedicated functionality. Controllers implement the interface exported by the managers. This way, the architecture is flexible, as only a controller needs to be exchanged if optimization of certain processing functionality is required. A dispatcher together with a configurable decision engine (decides, which messages to forward to a subsystem under which condition) implements the mediator pattern [22] and mediates between the subsystems, which are thus loosely coupled by keeping managers from referring to each other explicitly. This also lets us vary manager interaction independently, for e.g. incorporating new call-setup strategies. Each subsystem (manager/controller pair) operates asynchronously. The following subsystems (in the following we refer only to the managers) are identified:

- *Administration Manager* offers an administration interface used for local or remote administration. This can be used by a 3rd party service provider to configure the MAU using e.g. remote interfaces, or Corba.

- *Policy Manager* provides policy management functionality. It maintains a policy repository and is responsible for policy exchange with other MAUs, or service domain entities like SIP Servers.

- *Session Manager* is responsible for signalling with remote MAUs, media senders or receivers. It manages session objects and transcoding session state machines during session creation, adaptation of quality for established sessions and session teardown. The Session Manager typically has a session controller that implements a protocol similar to SIP [23]. Currently, we are extending our End-to-End negotiation protocol [24] to support MAUs.

- *Service Manager* performs registration, update and de-registration with respective Service Domain entities like SIP Registrars. It is also responsible for announcing already established transcoding sessions so that clients are able to join directly at a MAU.

- *Media Manager* offers media adaptation and transcoding services. It provides an interface to request, modify and teardown transcoding and adaptation sessions. It also provides a monitoring interface so that other subsystems can be informed about critical application and transport QoS parameters, current frame rate, packet loss or transcoding delay. It forwards configuration requests to the media controller that implements the media transcoding and adaptation process. Also, the media controller requests local resources from the dispatcher via the media manager for its media processing.

- *Local Resource Manager* is responsible for requesting, reconfiguring and releasing local resources like CPU. If the MAU runs out of CPU, the resource manager notifies the dispatcher, which then can invoke load-balancing.

- *Bandwidth Manager* is responsible for bandwidth contracts for local network interfaces and handles bandwidth reservation requests (e.g. based on RSVP [25]). It interacts (via the dispatcher) with Local Resource Manager to trigger configuration of local network interfaces and traffic control. Also, it issues reservation requests for network resources to the local service domain entities (like SIP Proxy) or bandwidth broker [26], depending on the reservation model within the domain the MAU is registered with.

In this paper, we focus on the media controller as it is responsible for stream processing, QoS control and implements the transcoding and adaptation functionality. The QoS awareness is achieved by providing a media manager that is MASA conform [1], [2] on top of the media controller. Thus, the media manager is used by other subsystems to request transcoding and adaptation services from the media controller and receives feedback on QoS delivery and coarsegranular adaptation requests, if the currently established QoS cannot be maintained. The overall MAU forms a Transcoding Broker using MASA terminology ([1]).

4.2. Media Controller Architecture

Figure 3 shows the architecture of the media controller,-which acts simultaneously as a virtual sink and source. It provides several InChannel to process packets received from a dedicated upstream node. Each InChannel has an association with several OutChannels that each transcode/adapt the stream for a dedicated downstream node, independently. In Figure 1, each MAU has one InChannel associated with two OutChannels. Adding another heterogeneous client to MAU 3 that wants to receive the video stream of sender A in a different format requires to create and configure another OutChannel and associate it with the InChannel that receives data from sender A via MAU1.

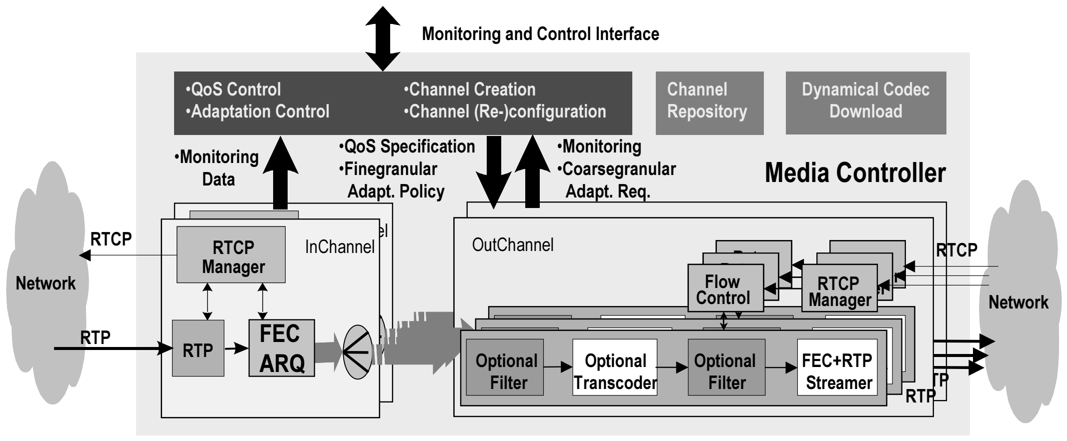

Fig. 3. Architecture of the media controller inside the media adaptation unit.

Each InChannel receives compressed media samples using RTP [20] and sends RTCP feedback to its upstream node. RTCP monitoring data (like round trip time and packet loss rate) and processing requirements are forwarded to a local QoS control entity that decides, if the current transport layer QoS (packet loss rate, delay, jitter) matches the QoS that was configured during InChannel creation by the media controller. The RTP packets are forwarded to a FEC/ ARQ module which initiates re-transmission of lost RTP packets according to [27] if the deadline for delivering the packet has not passed. The deadline is derived from the maximum tolerable MAU processing delay and the time needed for processing within an OutChannel. Alternatively, the FEC module tries to recover lost packets according to [28] or, if layered video is used according to [29]. The mechanisms used depend on the QoS requirements and the time constraints for recovering lost packets. If the QoS in the InChannel cannot be achieved, the Media Controller forwards an adaptation request via its control interface to the broker, which would then, based on QoS adaptation policies, notify the upstream node. This way, the sender might finally be notified and a trading algorithm [1] can decide at the sender to e.g. increase redundancy, change the access network or switch to another codec.

During initial channel creation, an OutChannel is associated with a proper InChannel for each downstream node. Each OutChannel receives information from the QoS control entity inside the media controller during channel creation and re-configuration about the downstream nodes capabilities (like media format and its configuration), application layer QoS parameters (like frame rate, frame size, quality of luminance plane,...) and maximum values for transport layer QoS parameters to restrict bandwidth and processing delay. For each frame the OutChannel accesses a list of RTP packets (provided by the associated InChannel), which belong to the current frame (we assume that the sender uses the same RTP timestamp for all packets of a given frame) and starts its processing cycle per frame. Once all OutChannels have finished their processing cycle, packets are destroyed from the InChannel. In each processing cycle, an OutChannel uses first a packet based filter, if possible, to adapt the quality based on application layer framing and packet dropping (see Section 3). This filtering process is very fast and is applicable to e.g. WaveVideo [17], [18], [19] streams. In this stage, the number of packets to be processed is reduced. This may lead also to a reduction in frame rate, frame size and visual quality, depending on filter configuration derived from application layer QoS parameters.

As the OutChannel knows the format of the incoming packets based on RTP payload types, it can decide on its own, if transcoding modules are necessary. It instantiates them, once the first packet from an InChannel is processed. After the transcoding step, another optional filter can be used that is able to understand the target format. Finally, a packetizer (not shown explicitly in the Figure) module segments the compressed and optionally filtered frame into RTP packets which the RTP streamer transmits to the downstream node (the final receiver or another MAU). The RTP streamer also implements FEC/ARQ mechanisms together with a RTP packet cache. The RTCP Manager in the OutChannel receives RTCP Receiver Reports which may contain implicit requests for retransmissions [27]. These requests are forwarded to the RTP Streamer that re-transmits them upon request from the RTP packet cache. Finally, the Flow Control module implements TCP-friendly rate control [30] and reconfigures the filter/transcoders with a new target data rate. When using WaveVideo, this results in a reconfiguration of the DRS-P [19] filter with a new value of BW_target. As DRS-P trades of bandwidth versus quality and there are many quality dimensions (frame rate, size and visual quality), each OutChannel is configured with a finegranular adaptation policy, that decides what quality dimensions to degrade first, if the actual bandwidth exceeds the target bandwidth [19]. If during operation a given OutChannel can not longer maintain the given target application QoS parameters, it issues an adaptation request via the control interface to the media manager which is finally processed by the broker. The broker decides to signal this adaptation request to the upstream node (if the problem was caused not by limited local resources) or associates more resources to the given channel.

The dynamical codec download entity within the media controller is responsible for extending the system. The channel repository holds a list of all InChannels and associated OutChannels together with the target QoS contract for each channel. Channel creation is triggered by signalling to the MAU, which instantiates a transcoding/adaptation service request via the control interface. The media controller creates a new In-/OutChannel pair or associates a new OutChannel to an already existing InChannel, and updates the channel repository.

4.3. Implementation Details

The filter modules that implement the filter algorithms [17], [18], [19] are part of our architecture at the media controller inside MAUs. Filter operation is thus completely controlled by the system itself and only coarse grained adaptation requests are forwarded to the higher layers so that a decision can be made to e.g. change the codec to a more error tolerant one. During the packet based filter operations specific segments of the compressed frame are dropped. In our implementation we transmit the information, how many segments (RTP packets) belong to a frame inside a RTP extension header. The receiver has to distinguish between filter operations, that explicitly drop packets to adapt the quality on demand and packet loss caused by transmission errors. Otherwise the receiver may try to compensate for the presumed packet loss by applying FEC/ARQ mechanisms. Since the media stream from the upstream node to the InChannel and the media stream from the OutChannel to the downstream node are transmitted in separate RTP sessions, the sequence numbers of the RTP packets generated by the OutChannel must be continuous. Additionally, we adjust the RTP extension headers of the packets in the OutChannel such that the number of segments belonging to the frame reflects the changes produced by the filter/transcoder.

During the whole processing of a media frame, we minimize packet copy operations to decrease processing time. For each incoming frame, the InChannel holds a list of pointers to individual (already FEC processed) RTP packets for the given frame. It passes this pointer list to each of its associated OutChannels without copying the packets. An OutChannel creates an object of a codec-specific class. This *Frame* object creates a second pointer list without copying the packets itself. Each pointer in this list points to the payload of an RTP packet. The filter modules then operate on this Frame object, i.e. only on the packet list. As the filter operations only require to delete pointers from the lists, the whole filter process does not require to copy the frame data itself. Therefore, filter operations can be performed in realtime. The filter module thus generates an already segmented frame. If transcoding is not necessary, the OutChannel only has to generate a new RTP header and insert it before the original payload for all remaining packets. If the frame shall be transcoded to another format, we first copy the payload of the indivual RTP packets for a given frame into a contiguous piece of memory. The transcoder then produces a frame with the new format. The OutChannel creates a second Frame object, which segments the transcoded frame and forwards it to the FEC module and the RTP streamer.

In our implementation, the transcoding module is based on the Video Compression Manager (VCM) built into the Win32 plattform to convert video frames from codec A to codec B. Without any optimization, this is achieved by two consecutive conversion steps. In the first step, the compressed frame is decompressed. During the second conversion step the uncompressed frame is re-compressed using codec B. These steps require that the appropriate VCM compressor/decompressor modules are available. As with all codec specific modules , the media controller can dynamically download and install compressor/decompressor modules during runtime, if necessary. For some combinations of codecs it is also possible to enhance the performance of the transcoding process by using special transformation algorithms. If intermediate processing steps allow for skipping parts of the decompression/compression pipeline, the performance can be significantly enhanced. One possible technique is to re-use parts of the motion vector information. Examples are published e.g. for H.263/MPEG-2 [31]. The media controller first determines the VCM compressor module, that is capable of performing a direct conversion from codec A to codec B in a single step. If such a compressor cannot be found, the media controller determines a decompressor for codec A and a compressor for codec B and performs the two-step conversion.

5. MEASUREMENT RESULTS

In this chapter, we present measurements and analyze the scalability of our approach. We distinguish between pure media stream adaptation based on packet dropping and transcoding.

5.1. Media Stream Adaptation

We were particularly interested in how many streams a MAU can process in parallel. We set-up a scenario, where an InChannel receives a WaveVideo packet stream and forwards it to an increasing number of OutChannels. Each OutChannel filters and streams the non dropped packets to a different downstream node. Thus, each client receives a filtered version of the original stream. The transcoder was running on an AMD Duron 700 MHz system at 256 MB RAM, Windows NT 4.0 (SP6). Figure 4 shows the total time to apply the filter module for each output stream to the aliens trailer (304x168, 15 fps). The minimum number of WaveVideo RTP packets per frame was 4, the maximum was 19 at an average of 11 packets/frame.

The color channel was removed and the size of the luminance channel was reduced by a factor of two. A new client was added each second. In total, 20 clients were served. Note that in this experiment we are only interested in the

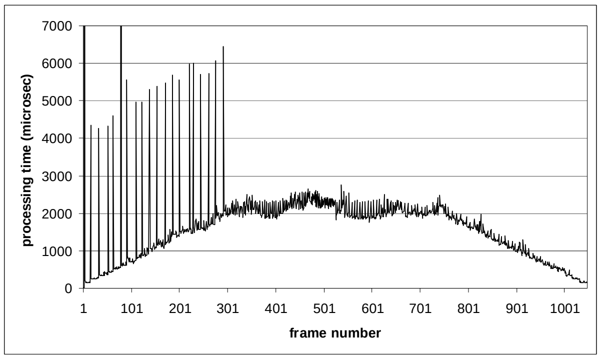

Fig. 4. Processing time at a MAU with increasing number of clients.

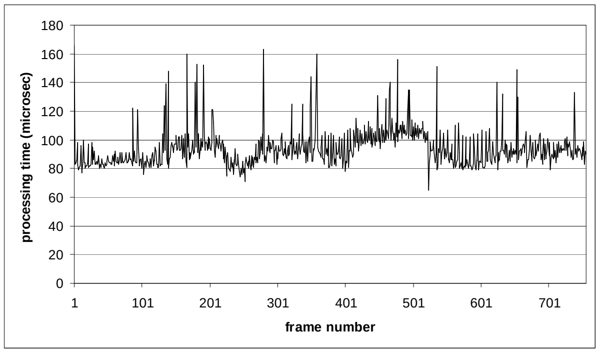

Fig. 5. Processing time at a MAU experienced by a single client

time needed for performing the filter operations to implement packet dropping based on application layer framing and our WaveVideo filters. This includes requesting the packet list from the InChannel, filtering the packet list and constructing a pointer list to all packets that remain after the filter operation. All these packets are subject to transmission to the downstream node. We measured the time at the InChannel between the point in time where the first request from an OutChannel to get the packet list arrived until the de-registration request issued by the last OutChannel. The packet list is forwarded to all OutChannels in a round-robin fashion and we measure total accumulated filter time for all channels.

We also denote the time necessary for processing an individual frame in Figure 5. In a real scenario, however, if each client received the same version of the movie, the filter operations would be applied only once and the filtered stream would be replicated to the receivers or an adapt request would be sent to the upstream node to forward a scaled down version. For the purpose of this analysis, the same filter operation was applied to each output stream. Otherwise, the measurement would not be comparable. The time in Figure 5 includes the construction of the pointer list that holds the packets for the given frame.

As can be seen from Figure 4, the time to filter a frame scales linearly with the number of clients added. This results in a linear increase in the delay observed by each client. Whenever a new client is added, an OutChannel is connected to the given InChannel. This also requires initialization of the processing modules in the OutChannel (like filter, RTP packetizer and streamer,...), but no transcoder was used. As each object is placed in a separate dynamic link library, multiple DLLs are loaded. Typically, between 1.5 and 2.5 ms are needed for this initialization, which results in the spikes that occur once the request for adding an outgoing channel is received. After frame 300, all channels have been added, the filters initialized and the stream is adapted and forwarded individually to 20 clients. After another 30 seconds (at frame 750), one OutChannel is removed each second so that at frame 1050, all OutChannels are disconnected. During the channel removal phase, a linear decrease in delay can be observed. Even if 20 clients are served, the additional delay due to filtering is below 3 ms (between frame 300 and 750).

5.2. Media Transcoding

In a second test scenario, we measured the processing time for converting a video stream from one codec to another. The MAU was running on an Intel Pentium-4 1.8 GHz system at 512 MB RAM, Windows 2000. Figure 6 shows the total processing time per frame needed for each OutChannel involved in this test scenario. In this scenario we used a sender terminal S1 with installed WaveVideo and DIVX codec [1] and three receivers. Receiver R1 and R3 decodes only WaveVideo, whereas receiver R2 decodes only DIVX. A MAU was used to adapt the video stream to fit the requirements of the receivers. During the whole test sequence the *matrix* trailer was sent to the MAU at a frame rate of 15 fps, CIF resolution. The trailer consisted of 450 frames, which were streamed permanently in a loop during the whole duration of the test sequence.

During the first period (frame 1-245), only R1 and R2 were participating in this session, and S1 sent the video stream to the InChannel of the MAU using WaveVideo codec. The InChannel was associated with two OutChannels during this period. OutChannel 1 sent the WaveVideo coded media stream to R1 without changing the media format. In parallel, OutChannel 2 transcoded the video stream to the DIVX format and sent it to R2. At frame number 246, R1 left the multimedia session. Since the upstream node of the MAU was also capable of transcoding the stream from WaveVideo to DIVX, the upstream node started transcoding at frame number 333. Thus the input format at the InChannel changed to DIVX at frame 333 and the MAU didn't have to perform transcoding for OutChannel 2 any more. At frame 743, R3 joined the multimedia session. Since R3 required a WaveVideo encoded stream and the InChannel received a DIVX coded stream at this time, OutChannel 3 performed transcoding from DIVX to WaveVideo format, and sent the transcoded stream to receiver R3.

[1]We used the version 4.12 from `http://www.projectmayo.com`

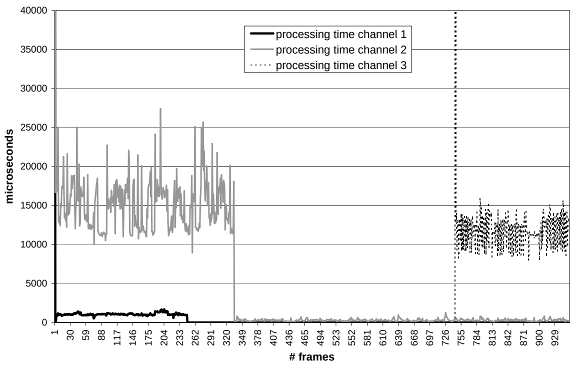

Fig. 6. Total Processing time for transcoding

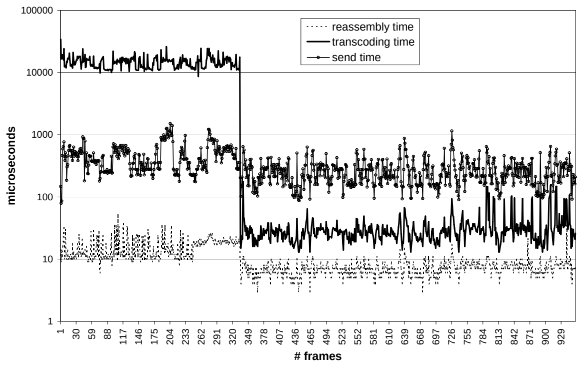

Fig. 7. Total Processing time for transcoding

The measured processing time (Figure 6) included requesting the pointer list from the InChannel, optionally reassembling the RTP packets, transcoding the frame into another compression format, creating and formatting the new RTP packets and streaming them to the downstream node. No FEC/ARQ was used to measure pure transcoding overhead. In the figure we can see, that the processing time for OutChannel 2 varied between 9 and 27 ms while performing the transcoding from WaveVideo to DIVX. This corresponds to a CPU utilization between 13 and 40%, which is significantly higher than using packet based filters for quality adaptation, as transcoding is a very time consuming process. Also, the transcoder was not optimized and converted from WaveVideo to DIVX by first decoding and then recoding. This requires a full motion vector search and additional delay at the receiver due to frame re-ordering of the DIVX. The processing time for OutChannel 1 was between 0.5 and 1.6 ms, because OutChannel 1 simply forwarded the packets after RTP header re-formatting. After frame number 333 the input format changed to DIVX, OutChannel 2 destroyed its transcoder and forwarded the DIVX packets. Here, that the processing time per frame varied between 0.1 and 1.2 ms at an average of 0.3 ms for OutChannel 2. Note, that the processing time needed by OutChannel 2 for only forwarding DIVX RTP packets was even less than the processing time needed by OutChannel 1 for forwarding WaveVideo RTP segmented packets. The reason was that we split compressed DIVX frames into RTP packets with a maximum length of 1500 bytes, whereas the WaveVideo packets were much smaller. Thus more WaveVideo packets were forwarded per frame.

At frame number 743, OutChannel 3 started the transcoding process from DIVX to WaveVideo. Processing time per frame varied between 8 and 16 milliseconds for OutChannel 3. The transcoding from DIVX to WaveVideo requires thus less CPU resources than the transcoding from WaveVideo to DIVX due to the expensive motion vector search of DIVX. WaveVideo in contrast only uses intra coded frames and inter-coded frames based on frame differencing [7]. Additionally we can see high spikes in the performance measurements whenever a new output channel was created. At frame 1, when OutChannel 1 and OutChannel 2 were created, the processing time is around 67 milliseconds for OutChannel 1 and around 103 milliseconds for OutChannel 2. At frame 743, when OutChannel 3 was created, the processing time for this channel is around 97 milliseconds. This includes the time for creating and initialising the OutChannel objects themselves as well as the time for initialising the transcoding process. In comparison to the spikes which occured in figure 4 whenever a new OutChannel was created (4 milliseconds), the measured processing time at the spikes in figure 6 is much higher. In the test scenario shown in figure 4, there was no need to initialize any compressor/decompressor modules as we were only interested in adaptation. Therefore, we can see that most of the processing time at the spikes in figure 6 was used for initializing the compressor/decompressor modules. In an enhanced implementation, this time can be reduced to almost zero by creating and initializing compressor/decompressor modules for different combinations of codecs in advance and only creating references during Channel initialization.

Figure 7 shows in more detail the time needed for the individual processing steps for OutChannel 2 using a logarithmic scale. *Reassembly time* denotes the time needed for reconstructing the original frame from the RTP packets. *Transcoding time* denotes the time needed for the two-step transcoding process. *Send time* is the time needed for segmenting the frame into RTP packets and sending them to the downstream node. Reassembly time is almost neglectable (around 10 microseconds). Nevertheless, one can see that the reassembly time drops after frame 333. This is due to the fact that the DIVX frames are segmented into fewer RTP packets than the WaveVideo frames. The send time is also below 1 millisecond. Therefore, the total processing time for an OutChannel depends primarily on the time needed for the transcoding process.

Figure 8 shows the bandwidth of the video stream before and after the processing in the MAU. As OutChannel 1 never performed any transcoding at all, the bandwidth curve for OutChannel 1 is identical to the bandwidth curve of the InChannel. The bandwidth of the outgoing video streams 2 and 3 differ from the bandwidth of the incoming video

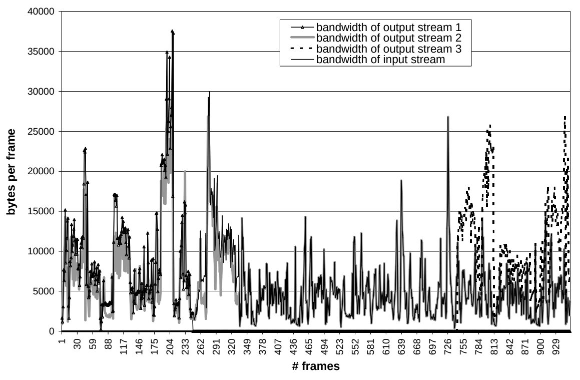

Fig. 8. Bandwidth of the video stream before and after transcoding

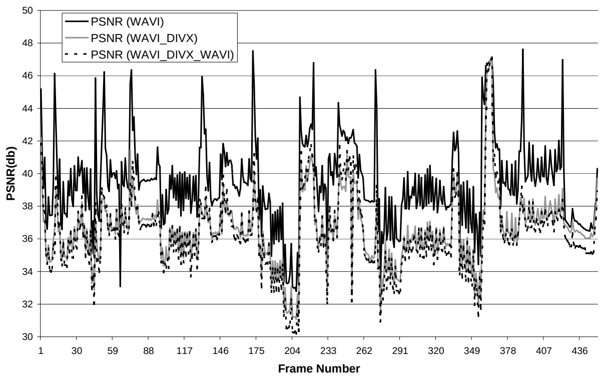

Fig. 9. PSNR of compressed version (WAVI compared to transcoded versions).

stream only during the periods when the respective OutChannels performed transcoding. The bandwidth of the input (DIVX) stream varies between 1000 and 14000 bytes per frame from picture 743 to 954 (note that the frame rate was constant at 15 fps), whereas the frame sizes of stream 3 (WaveVideo) varied between 2000 and 26000 bytes.

Figure 9 shows the PSNR (luminance channel) of the Wavevideo compressed stream (WAVI), which varied between 32.77 and 47.62 dB at an average of 39.4 dB experienced by receiver 1. We measured PSNR after the transcoding process within the MAU and assumed that no packets were lost during streaming. It also shows the PSNR of the transcoded version (WAVI_DIVX), where the format of receiver 2 was DIVX. Here, the quality varied between 31.07 and 47.04 dB at an average of 36.55 dB. Finally, the format of receiver 3 was WAVI and the MAU transcoded the WAVI to DIVX transcoded stream to WAVI again (WAVI_DIVX_WAVI) which resulted in an average PSNR of 36.29 dB.

6. SUMMARY AND CONCLUSION

In this paper, we have presented the design of a generic media adaptation unit that is able to transcode live video streams and adapt their quality on demand using per packet filtering. The concept of filtering compressed media streams becomes interesting if filter operations can be performed without significant overhead. Our evaluation showed that filter algorithms can be applied in real-time if proper application layer framing is used. Media adaptation units based on such packet dropping mechanisms are scalable with respect to number of clients served. We have provided support for clients with heterogeneous capabilities by adding transcoder modules inside the media adaptation units. Transcoding of live video streams can be achieved on current off-the-shelf hardware only with a limited number of supported clients. Using multi-processor machines or special hardware support for transcoding should lead to a dramatic improve in performance.

The most important drawback is that filtering is not suitable to address scenarios that require loss-less transmission of the media streams. By design, filtering is a lossy concept to reduce data rate and thus quality. However, in our architecture we coordinate packet filtering with error detection and FEC/ARQ techniques to cope with packet loss due to pure transmission errors. This is important as important packets that survived the filtering process (e.g. containing low frequency information) may get lost in the network.

We are currently extending our media adaptation units to several other areas. Firstly, we are looking at audio transcoding and first experimental results are very promising. We believe that we will be able to support between 50 and 100 concurrent transcoding sessions from e.g. PCM to GSM on standard PCs. Processing delay might be a concern for real-time audio communication. Using special filters that discard all video packets and forward only audio packets, multiplexed system streams can be demultiplexed. This is useful, if certain receivers (like mobile phones) are not interested in the video stream at all. Our framework is so flexible, that only a filter module has to be implemented that works together with the proper RTP packetization scheme. Also, the semantical filter algorithms are implemented above the socket layer, which makes it suitable to be used across a number of different network technologies. For example, support for ATM networks can easily be provided. Finally, we will integrate the FEC and ARQ modules and test them in a real heterogeneous UMTS testbed together with handovers to Wireless LAN using multi-homed terminals.

7. ACKNOWLEDGEMENT

The work was partially funded by Siemens AG, Research and Concepts Department of ICM Networks, in the scope of the MASA Network project. Additional funds were provided by the DFG within the AKOM framework. The authors would like to thank their colleagues from Siemens AG and NEC Europe Ltd., Heidelberg, for their support. Finally, we are grateful for the reviewers comments that helped to improve the quality of the paper.

8. REFERENCES

[1] H. Hartenstein, A. Schrader, A. Kassler, M. Krautgaertner, and C. Niedermeier, "High Quality Mobile Communication," *Proc. of the KIVS 2001*, March 2001.

[2] A. Kassler, C. Kücherer, and A. Schrader, "Adaptive Wavelet Video Filtering," in *Proc. 2nd International Workshop on Quality of future Internet Services (QofIS)*, (Coimbra, Portugal), Sep 2001.

[3] J. Pasquale, G. Polyzos, and V. Kompella, "*The Multimedia Multicasting Problem*," tech. rep., University of California, San Diego, CS93-313, 1993.

[4] M. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley, "Periodic Broadcast and Patching Services - Implementation, Measurements, and Analysis in an Internet Streaming Video Testbed," *Proc. of ACM Multimedia Systems 2001*, October 2001.

[5] C. Griwodz, M. Zink, M. Liepert, and G. Steinmetz, "Internet VoD Cache Server Design," *ACM Multimedia 99*, pp. 123–126, October 1999.

[6] L. Teixeira and M. Martins, "Video Compression: The MPEG Standards," *Proceedings ECMAST 96*, pp. 615–634, May 1996.

[7] G. Fankhauser, M. Dasen, N. Weiler, B. Plattner, and B. Stiller, "WaveVideo - An Integrated Approach to Adaptive Wireless Video," *ACM Monet, Special Issue on Adaptive Mobile Networking and Computing*, vol. 4, no. 4, 1999.

[8] L. Mathy and O. Bonaventure, "QoS Negotiation for Multicast Communications," *Proceedings of the COST 237 Workshop on Multimedia Transport and Teleservices, Vienna, Austria*, November 1994.

[9] H. Wolf, K. Froitzheim, and M. Weber, "Interactive Video and Remote Control via the World Wide Web," *Proceedings of IDMS 1996*, pp. 227–243, March 1996.

[10] W. Strayer, S. Timothy, and R. E. Cline, "An Object-Oriented Implementation of the Xpress Transfer Protocol," *Proceedings of the Second International Workshop on Advanced Communications and Applications for High Speed Networks, Heidelberg, Germany*, September 1994.

[11] N. Yeadon, F. Garcia, D. Hutchison, and D. Shepherd, "Filters: QoS Support Mechanisms for Multipeer Communications," *IEEE Journal on Selected Areas in Computing (JSAC) special issue on Distributed Multimedia Systems and Technology*, vol. 14, pp. 1245–1262, 1996. San Jose, CA.

[12] J. Pasquale, G. Polyzos, E. Anderson, and V. Kompella, "Filter Propagation in Dissemination Trees: Trading Off Bandwidth and Processing in Continuous Media Networks," *Proc. of 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 269–278, 1993.

[13] S. McCanne, M. Vetterli, and V. Jacobson, "Low Complexity Video Coding for Receiver-Driven Layered Multicast," *IEEE Journal on Selected Areas in Computing (JSAC)*, vol. 16, no. 6, pp. 983–1001, 1997.

[14] E. Amir and S. McCanne and M. Verterli, "A layered DCT coder for Internet Video," in *Proceedings of the IEEE Conference on Image Processing ICIP'96*, September 2001.

[15] F. Raspall, C. Kuhmuench, A. Banchs, F. Pelizza, and S. Sallent, "Study of Packet Dropping Policies on Layered Video." *Proc. of the Workshop on Packet Video*, Korea, 2001.

[16] D. Clarke and D. Tennenhouse, "Architectural Consideration for a New Generation of Protocols," *Proc. ACM SIGCOMM*, pp. 200–208, September 1990. Philadelphia, PA.

[17] A. Kassler, A. Neubeck, and P. Schulthess, "Classification and Evaluation of Filters for Wavelet Coded Videostreams," *Signal Processing: Image Communication*, vol. 16, pp. 795–807, May 2001.

[18] A. Kassler, A. Neubeck, and P. Schulthess, "Real-time Filtering of Wavelet Coded Videostreams for Meeting QoS Constraints and User Priorities," *Proc. of Packet Video Workshop*, Mai 2000.

[19] A. Kassler and A. Neubeck, "Self Learning Video Filters for Wavelet Coded Videostreams," *Proceedings of the Int. Conf. of Image Processing (ICIP2000)*, September 2000.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications." *IETF RFC 1880*, January 1996.

[21] G. Keesman, "Transcoding of MPEG bitstreams," in *Signal Processing: Image Communication*, vol. 8, pp. 481–500, 1996.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[23] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "Session Initiation Protocol (SIP)," *IETF RFC 2543*, March 1999.

[24] A. Kassler et al., "Enabling Mobile Heterogeneous Networking Environments With End-to-End User Perceived QoS - The BRAIN vision and the MIND approach," *Proc. of the European Wireless Conference*, pp. 503–509, February 2002. Florence, Italy.

[25] L. Zhang, S. Deering, D. Estrin, S. Schenker, and D. Zappala, "RSVP: A New Resource Reservation Protocol," *IEEE Network Magazine*, September 1993.

[26] A. Terzis, L. Wang, J. Ogawa, and L. Zhang, "A two-tier resource management model for the internet." *in Proceedings of Global Internet*, December 1999.

[27] A. Miyazaki et al., "RTP Payload Format to Enable Multiple Selective Retransmission." *Internet Engineering Task Force, Work in Progress*, November 2001.

[28] J. Rosenberg et al., "An RTP Payload Format for Generic Forward Error Correction." *Internet Engineering Task Force RFC 2733*, December 1999.

[29] G. Liebl et al., "An RTP Payload Format for Erasure-Resilient Transmission of Progressive Multimedia Streams." *Internet Engineering Task Force, Work in Progress*, February 2001.

[30] S. Ramesh and I. Rhee, "Issues in Model-Based Flow Control," Tech. Rep. *TR-99-15*, Department of Computer Science, North Carolina State University, 1 1999.

[31] N. Feamster and S. Wee, "An MPEG-2 to H.263 Transcoder," *SPIE Voice, Video, and Data Communications Conference*, September 1999. Boston, MA.