Homer Conferencing – a Multimedia Test Bed for Various Experiments and Measurements

Thomas Volkert, Andreas Mitschele-Thiel
Integrated Communication Systems Group
Ilmenau University of Technology
Ilmenau, Germany
{thomas.volkert, mitsch}@tu-ilmenau.de

Martin Becke, Erwin P. Rathgeb
Computer Networking Technology Group
University of Duisburg-Essen
Essen, Germany
{martin.becke, erwin.rathgeb}@iem.uni-due.de

Abstract— **Transferring multimedia data from and to customers is one of the biggest challenges of current network engineering. A key challenge, researchers are confronted with, is to understand what kind of real-time support is needed, what the network could provide to the application and how the application could utilize codecs, quality settings and transmission protocols in order to assure the service. In this paper we introduce the open source software Homer-Conferencing (HC) as multimedia test bed for research purposes. Our approach was developed during the last four years. The main goal of our work is to allow researchers implementing deep analysis of multiple kinds of multimedia traffic. For this purpose, exchangeable codecs and adjustable transmission qualities can be used in a video conferencing application. In this paper, we describe the architecture of HC and highlight based on selected experiments the benefits the HC system provides for experiments and measurements in currently used networks as well as future Internet platforms.**

Keywords—Conferencing, case study, QoS, transmission requirements, multimedia streaming, peer-to-peer, Future Internet.

I. INTRODUCTION

Today's Internet is based on the well-known "Open Systems Interconnection" (OSI) reference model [1]. It works well for known services such as mail or simple file transfer. However, in the recent years coping with the requirements, needed by applications, became an increasingly challenging task.

In fact, the robust and longtime used current network architecture has multiple drawbacks [2], such as the lack of efficient support of popular services of today, e.g. Real-Time (RT) multimedia transmissions. Such services have variable requirements for transmissions. However, an application requirements-driven combination of functional blocks is not possible. Currently, the application can select between predefined static function-bundle-sets, which are provided by existing protocols of different OSI layers. Special protocol functions cannot be addressed explicitly or combined to a new chain of functional blocks [3]. For example, the currently used network stack is not able to support an unreliable, ordered and encrypted transmission, although multiple protocols exist which support these functions separately. However, this combination of functional blocks could be desirable for a secured transmission of a video stream. Besides the lack of support for a flexible function combination, there also exist redundant implemented functions. In today's networks, protocols are simultaneously applied which provide similar or the same special functional blocks, e.g., load sharing and congestion control on the transport layer [4, 5]. But there is no common interface, which allows the addressing and the combination of such functional blocks in a general way. Briefly summarized, there is no functional block base for providing a service as demanded by RT applications.

In general, the transport of application data over a communication network like the Internet is definitely a complex task, and the complexity is increasing by using interim solutions such as shim layers, cross layer approaches and other optimizations in order to tackle the needs of recent and future applications. There is a need for supporting application requirements and fulfill them by instantiating functional blocks. The demands are very fine granular and the needed functions could be found not only on the transport level but also across the layers when using session handling, security, routing, load sharing or QoS. One goal of the current research is to figure out, which requirement descriptions are needed and if and how the network could provide this. One of the examples we want to highlight in this context is the use of RT communication where "best effort" is definitely not enough. In fact, QoS requirements have to be fulfilled [6], such as a maximum end-to-end delay or a minimal provided end-to-end data rate. In this context, the research community of today needs not only platforms like GENI, G-Lab, F-Net or Planet Lab, but also tools and demonstrators which enable measurements for comparing studies or implementing proof-of-concept scenarios. In this work we want to contribute to this research area by introducing our software approach, we denote as Homer Conferencing (HC) system [7]. Our solution enables testing new protocols and network design principles by providing a common multimedia communication test platform designed for RT communication.

In this paper, we first give an overview on the main design goals and the adaptive architecture of our solution. Following this,, we demonstrate within the evaluation part of this paper the benefits of our solution in a specific use case where HC allows using different codecs and further describe how HC helps understanding the application needs, its behavior and observe the resulting data streams.

This work was funded by the German Federal Ministry of Education and Research of the Federal Republic of Germany through the project "G-Lab_FoG" (support code 01BK0935) and the special-interest-group "Multipath". Both are part of the German Lab (http://www.german-lab.de).

II. DESIGN PRINCIPLES

Starting the development in 2008, HC was built up as an alternative to existing open source tools like "Ekiga" [8] and "Linphone" [9]. Since the start of development, the focus of HC was on separate logical transport streams. The HC software was designed to support separate multimedia streams with explicitly given transmission requirements per stream, something that was not possible with existing software and is still difficult to implement by the help of alternative software of today. Moreover, the software supports recording of measurements per logical stream and not only per connection or session. Based on the support of separate stream transmissions, HC supports the development and performance measurements of next generation networks or Future Internet approaches. In general, the goal was also to create a tool which allows a deeper look into RT transmissions and application behavior, e.g. by using different static and adaptive audio and video codecs.

In order to study the application behavior in complex scenarios, the software also supports (RT) video conferencing. This scenario requires the transmission of different kinds of application data: control data for conference management, and chat messages, audio and video (A/V) streams for participant interaction. Every kind of application data has its own requirements for a transmission. For example, control data has to be transmitted in a reliable manner while multimedia streams can tolerate packet loss at a low rate. In addition, the packets of all streams have to be transmitted in an ordered way.

An additional design principle for the HC development was to provide all desired application services in a decentralized peer-to-peer (P2P) operation mode. For this purposes, the "Session Initiation Protocol" (SIP) [10] and the "Session Description Protocol" (SDP) [11] were selected. They can be used in centralized as well as decentralized mode. Therefore, each conference session in HC is controlled based on the SIP protocol. With each conference session an audio and a video stream in bidirectional directions can exist. All of these streams have several media parameters, which have to be acknowledged by both session participants. In HC, the media parameters hand-shake is implemented based on the SDP protocol which here is implemented as sub protocol of SIP. HC applies SIP as well as SDP by the help of the library "Sofia-Sip" [12]. The internally used signaling messages implement a request-response protocol, which is similar to other XML-based service protocols [13]. As a result of this, each signal message of SIP/SDP consists only of human readable elements, which is beneficial when debugging the signaling processes. Moreover, the general concept behind SIP and SDP was kept rather simple in comparison to alternative approaches. For example, H.323 [14] represents such an alternative solution. But its main drawback is its complexity, caused by the need of many sub protocols for operation, such as H.225, H.245, H.450 and H.235. Moreover, this solution is based on binary encoded signaling messages, which hamper fast prototype development.

HC uses a generalized signaling interface towards the network in order to abstract the specific network stack implementation. This is implemented by the "network API" (NAPI). NAPI was inspired by the "G-Lab API" [15], which is one of the results of the work within the G-Lab Project [16]. The design of this API allows an easy integration of existing protocols by the well-known de-facto standard Berkeley sockets but also addresses Future Internet approaches like FoG [17]. As a result of this, HC handles data streams independently from the current network stack implementation behind the NAPI. By the help of NAPI, HC is able to define different kinds of requirements per stream. The current software release [7] allows adjusting transmission requirements based on user inputs. For example, the user is able to define a maximum end-to-end delay and a minimum end-to-end data rate. Additionally, functional blocks, e.g., reliable transmission, can be addressed by the help of transmission requirements.

A list of (desired and) implemented features is given in the following:

- cross platform development: Windows, Linux, OSX
- video conferencing in real-time via P2P transmissions
- video codecs: h261/h263/h264, mpeg1/2/4, theora
- audio codecs: mp3, g.711, pcm16
- streaming/recording/playback of A/V data
- screenshots of received video streams
- support for UDP, TCP, UDP-Lite, (CMT-)SCTP
- support for IPv4, IPv6
- adjustable A/V quality for streaming
- selectable transmission requirements for streaming

III. SOFTWARE ARCHITECTURE

The HC software consists of seven main software components, depicted in Table 1. The first one is the main program binary, which includes the graphical user interface (GUI). It is based on six additional software components, which are implemented by separate libraries.

Component	Responsibilities
GUI (program binary)	User interface for interaction with user, provides needed graphical views and controls
Multimedia (library)	A/V hardware access, A/V streams encoding/decoding, RTP packet processing
Conference (library)	SIP based conference management, SDP packet processing
SoundOutput (library)	Hardware access for sound playback
Monitor (library)	Observation of threads, measurement of data flows and generating of packet statistics
NAPI (library)	Network-API: abstracted Berkeley sockets, multiplexing between simultaneously available socket implementations
Base (library)	Operating system abstraction for Windows, Linux and Mac OSX

Table 1: Software components in HC

In order to support future extensions and modifications without the need to modify the entire software, HC is not only split into seven software components, but each software component consists of several sub modules. Furthermore, the "model view controller" (MVC) software pattern is applied to split the software into a graphical user interface (GUI) as

frontend and A/V processing (including sound output) as well as conference management as backend components. Frontend and backend instances run in a separate process context. The software components "NAPI" and "Base" implement separate abstraction layers for network access and operating system access, respectively. The latter allows to neglect any operating system specifics within higher software components, e.g., multi-threading support. Orthogonally to the already mentioned software components, a "Monitor" component is integrated. It is responsible for creating statistics about threads and data streams existing in HC. Based on the included software sensors and a dedicated graphical view, each data stream is observed by the monitor component and accumulated static values can be shown in the GUI.

In the following sections an overview of the possible data flows during A/V processing is given. Fig. 1 and 2 show the involved sub modules of the component "Multimedia". It distinguishes between media sources and media sinks. The multimedia processing uses the library "ffmpeg" [18].

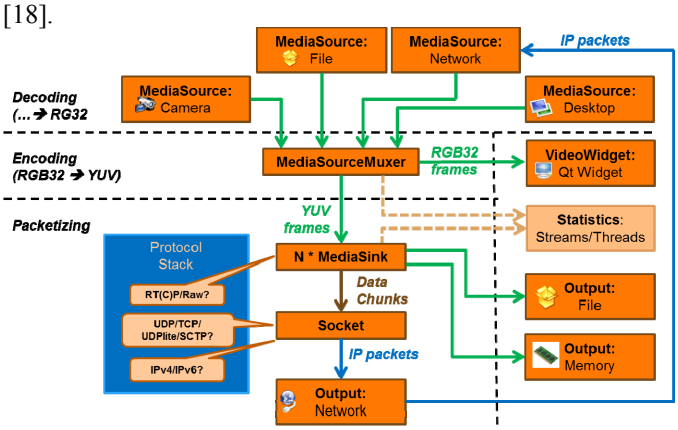

Figure 1: Possible data flows during video processing

A video source can be a connected device (e.g. a camera) for video capturing, a local file, the local desktop screen or a network source. Each of these video source modules is responsible for decoding the grabbed video stream from its individual format to RGB32 frames in order to allow a direct presentation in a video widget, consisting of a window on the desktop screen. Each RGB32 based video stream can be used as input for a "MediaSourceMuxer" (short: muxer) module. It is used for multiplexing between different media sources and encodes an incoming video stream according to a defined format, which can be selected via GUI. From the outside, a media muxer module can also be used as a simple media source because its interface is only an extension of the one from basic media sources. However, the extended functions of the muxer enable distributing each created video stream to different assigned media sinks. They can represent a local file, a memory area (for further processing) or a network destination. Each media sink module supports the "Real-time Transmission Protocol" (RTP) [19] in order to add timestamps and sequence numbers per data chunk. They are used at receiver side to detect packet loss and allow continuous video playback. Moreover, the additional RTP can be used for synchronization with audio playback in order to allow a maximum of quality of experience (QoE) for the user at the receiver side. In order to have a feedback about the quality of the transmission at both sides the "Real-Time Transport Control Protocol" (RTCP) [19] is also supported. It provides periodic reports from both sender and receiver, and allows them to compare the sent and received data in order to judge the quality of the transmission. This could be used for codec adaptions on the sender side in order to improve the QoE.

The network abstraction layer NAPI and its included default backend implementation for using Berkeley sockets are able to transmit an A/V stream based on one of the following layer 4 protocols: TCP, UDP, UDPLite [20] and SCTP [21]. In addition to the variation for layer 4 implementation, both IPv4 and IPv6 can be used as layer 3 protocol. HC does not necessarily need IPv4 for operation and, therefore, can also be used after the complete transition of the Internet from IPv4 to IPv6. The overall possibilities are limited by the capabilities of the underlying operating system and the supported socket types. Moreover, there also exist approaches for stream transmission based on Future Internet approaches [22-24].

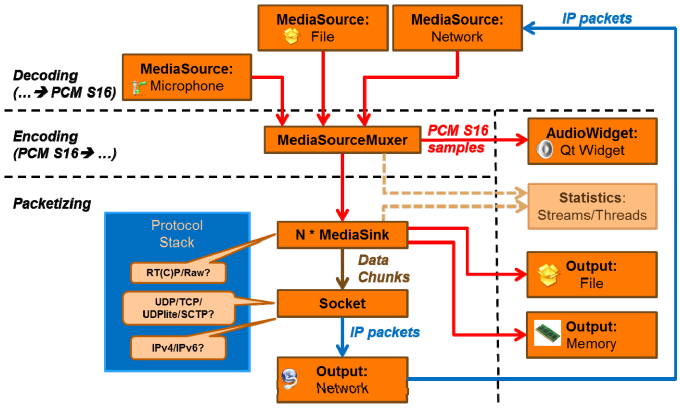

Figure 2: Possible data flows during audio processing

Besides the possibilities for video processing, HC also supports similar functions for audio processing. Fig. 2 gives an overview about this. As a result of the similar module structure, HC can easily be used to stream entire movies by using the media framework (from the multimedia component) to process both audio and video data from the same local file. Moreover, for audio streams HC supports the same set of transport and network protocols as for video streams. Therefore, a movie can be transmitted through any kind of network stack which is implemented behind the used network abstraction layer NAPI.

In order to allow an acoustic observation of processed audio streams, the muxer is able to output each stream in the standard audio playback format called "PCM16 stereo", which uses 16 bits per sample to quantify audio signals. This enables fast prototyping and comparison of protocols since the actual transmission quality can be determined qualitatively by such a RT playback of received audio streams.

IV. Application Examples

In the following examples we describe how HC can be used in Linux in order to capture a video from a connected webcam, transcode it via the H.263 codec and send the resulting video stream via IPv6 to port 5004 of the local machine.

In Example 1 we use the Linux API "Video4Linux2" (V4L2) for camera access. This part is encapsulated by the module "MediaSourceV4L2", which is automatically called by the muxer module each time Step 4 is processed. This step is executed periodically by the GUI while the stream is running.

Step 1: **initialize the needed objects**

```
MediaSourceMuxer *mux;
MediaSourceV4L2 *v4l2;
v4l2 = new MediaSourceV4L2();
mux = new MediaSourceMuxer();
mux ->RegisterMediaSource(v4l2);
mux ->SetOutputStream("H.263", ...);
```

Step 2: **start camera access via V4L2**

```
mux ->OpenVideoGrabDevice(...);
```

Step 3: **assign network sink**

```
mux ->RegisterMediaSink(,,::1", 5004, RTP_UDP);
```

Step 4: **process one single frame**

```
mux ->GrabChunk(...);
```

Example 1: Video capturing and distribution

In the second example we show how the video stream of example 1 can be received from the network based on the multimedia component. The proposed source code describes in principle the process and abstracts some implementation details. Compared to Step 4 of Example 1, Step 2 grabs a video frame from the network. This step also has to be executed by the GUI. According to the first example, Step 2 has to be repeated for each frame. As described in Section II, the two given examples can be modified in order to use - besides the standard RTP/UDP configuration - protocols like TCP, UDP-Lite and SCTP. The hereby optimal settings depend on the application scenario and which protocol features are needed during the A/V stream transport in order to provide the desired trade-off between QoE and network resource usage. Different settings were already used in [25]. For such scenarios HC supports the selection of different transmission requirements, which are signaled through the network abstraction layer NAPI towards the actually used network stack implementation. These requirements influence the behavior of the network stack behind the NAPI interface depending on the stack's capabilities.

Step 1: **initialize the needed objects**

```
MediaSourceNet *net;
net = new MediaSourceNet("::", 5004, RTP_UDP);
net ->SetInputStream("H.263", ...);
```

Step 2: **process one single frame**

```
net ->GrabChunk(...);
```

Example 2: Video reception

For observing the behavior of each data stream in HC, the GUI includes two table views dedicated to audio and video streams. They depict the following statistic values per processed stream:

- Min./max./average packet size
- Overall size of transmitted data
- Counter of transmitted packets
- Counter of lost packets
- Used transport/network protocol
- Current/overall data rate

For determining the continuous behavior of a processed stream over time, the GUI additionally enables exporting the entire set of measured data rate values to disc. For example, this is useful to determine data rates, which are caused in the network by the data sending HC instance. Moreover, this can be used to judge the quality and time behavior of a reliable transmission by comparing the values at receiver side with those from the sender side.

V. Experiments and Measurement Results

In the following, we describe a case study and measurement results describing the time-continuous data rates, which are acquired from the network resources during video transmissions. We have concentrated on video transmissions because their impacts on available network resources are bigger than those of audio streams. This fact is caused by the generally higher data rates and the used variable bit rate generated by a compressing video codec. As a result of this, we expected that the measurements show a varying transmission data rate and also a varying amount of acquired network capacities. To study the actual behavior and derive the expected range of the resulting data rate of such a video stream, we have used the proposed application examples of Section IV. In the existing related work, we were not able to find any data rate measurements, which applied both an old video codec like H.261 and a recent one like H.264, in a comparative study describing the throughput based on a real test bed. For our study we measured all data rates of video streams by using the code base of HC [7] without modifications. In order to measure the actual data rates, we used the included stream statistic outputs from the GUI, described in Section IV.

In general, we have created a measurement environment, which is representative for today's networks and which allows future extension. Therefore, all measurements were based on the same freely available animated movie [26] in order to have a publicly available data reference. Moreover, we used only well-known video conferencing codecs: H.261, H.263, H.264 and MPEG1/2/4 with different video resolutions and stream qualities. According to page limitations we present an excerpt of these results. In this paper, we focus on a fixed input video resolution of 854 * 480 pixels from the movie file and an output resolution of 176 * 144 pixels. The latter is created by the re-encoding process within the muxer module. Furthermore, we have adjusted the range for the video quantizer scale [18] to values between 1 and 24, which corresponds to a video quality setting of 10% within the HC

software. This setting has direct influence on the resulting bit rate of an outgoing video stream. Within each generated stream, every 18 frames a key frame was used and we have limited the size of the resulting output packets in the GUI of HC to 1280 bytes, which is the MTU size for an IPv6 based link [27]. All measured values were collected at the sender side after the video transcoding process and before the resulting packetized stream of IP packets is send to the network. As a result of this, we have measured in our experiments the actual data rate, the stream would acquire from the network without any kind of stream buffering, e.g. by socket buffers or additional application queues. Figure 3 shows the measured curves for the selected video codecs: H.261, H.263 and H.264.

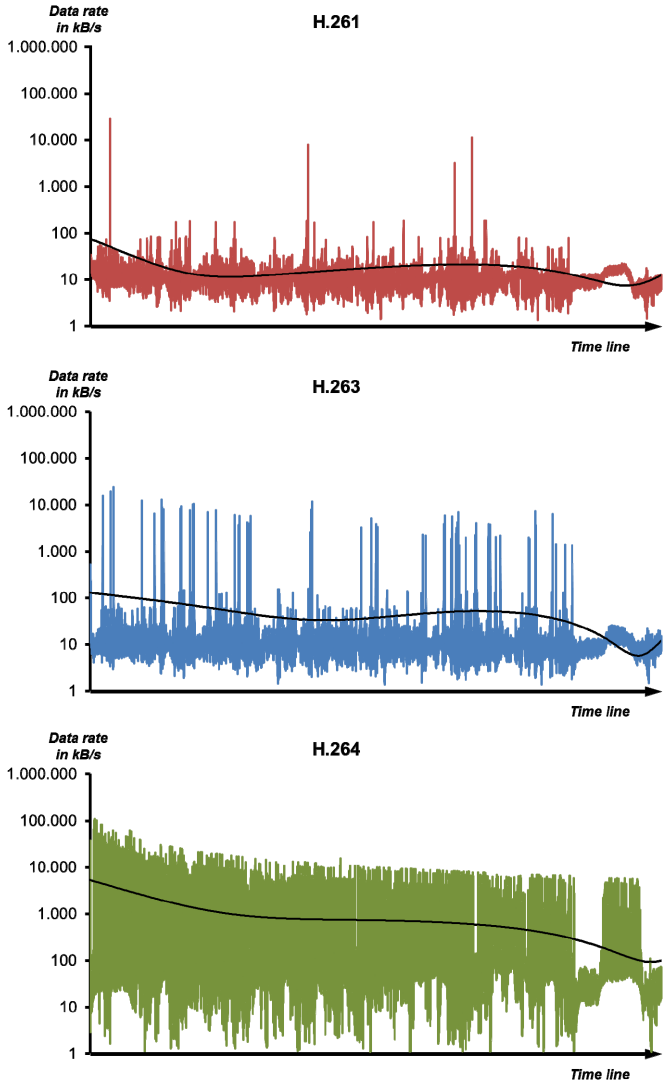

Figure 3: Resulting data rates during video streaming

Every measured value was determined when a packet left the video processing chain. The frame rate of the input video was 24 frames per second and the output packet rate varied depending on the applied packet fragmentation. The colored curves of Figure 3 show very clearly the scene changes, especially at the end of the movie. The resulting behavior changes of the outgoing data rate can be seen as data peaks in Figure 3. Especially in the case of the H.264 codec these peaks appeared very often but they lasted only for short time periods. Because of this, the resulting average is much lower than the peaks are. Moreover, the curves show that the codecs H.261 and H.263 have a similar behavior, while the H.264 codec causes a much higher data rate for the same input data. In the depicted figures, we have added black curves in order to show the data rate trend for the entire movie length of 10 minutes.

In Fig. 4 the received video qualities for the 3 measured video codec are depicted. The pictures show that the output quality increases from H.261 to H.264, especially visible at the clouds in Fig. 4.

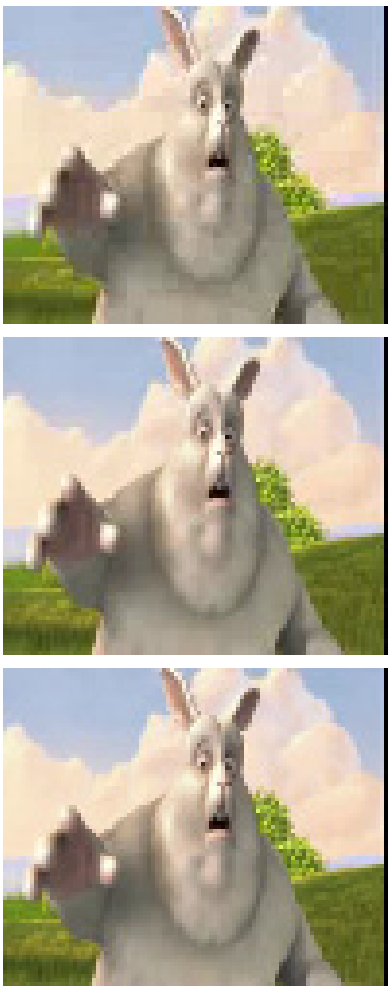

Figure 4: Received H.261, H.263 and H.264 picture

Compared to Fig. 3, it is clear that H.264 requires the most network resources but it delivers the best QoE.

Table 2 shows measurement statistics. The most interesting values are the high maximum peaks, which appear for all video codecs. They represent the actual caused data rate before any kind of buffering is applied. In opposite to this, the average data rates can be interpreted as approximation of the data rates which would be acquired from the network.

Name of value	Used video codec		
	H.261	*H.263*	*H.264*
Overall Packets	15.939	16.916	26.694
Average packet size	402 bytes	369 bytes	771 bytes
Overall data	~6,12 MB	~5,96 MB	~19,65 MB
Average data rate	~10,59 kB/s	~10,23 kB/s	~33,78 kB/s
Max. data rate	~28,22 MB/s	~23,88 MB/s	~108,23 MB/s
RTP packet header	12 bytes	12 bytes	12 bytes
RTP payload header	4 bytes	2 bytes	1-2 bytes
Absolute overall RTP overhead	~249,05 kB	~231,27 kB	~364,96 kB
Relative overall RTP overhead	3,97 %	3,79 %	1,81 %

Table 2: Measurement statistics

VI. CONCLUSION AND FUTURE WORK

In this paper we have proposed Homer Conferencing (HC) as open source software for experiments and measurements for understanding multimedia applications, their requirements to networks, and the behavior of multimedia streams. Since the start of the software development in 2008 the software was developed as an end user and research software. The latter is shown in this paper by demonstrating how HC can be used to accumulate measurement statistics of multimedia streams in a real test bed in order to have a base for comparative experimental studies of transport technologies. HC's feasibility to be used for experiments on different protocols layers in current as well as future network architectures has been documented in [22, 23, 25, 28].

In this paper we have evaluated the behavior of video streams and their requirements for network transmission. In the future we plan to use these values as input for an automatic requirements selection in order to improve network usage efficiency based on approaches described in [23]. Additionally, we plan to enable more specific functions of architectural approaches, features and extensions of selected protocols and codecs, e.g., SCTP, DCCP, IPv6 or QoS signaling. We plan to make the HC system more comfortable for end-users in order to allow surveys from distributed measurements.

ACKNOWLEDGMENT

The authors would like to thank the anonymous testers of the HC system for their constructive comments and suggestions for improving the overall system quality. In this context we address especially the research project G-Lab [16], which supported the development and allowed intensive testing of the HC architecture on different network architecture approaches.

Furthermore we would like to thank the members of the ICS Group at the University of Technology in Ilmenau, Germany, for valuable discussions and feedback.

REFERENCES

[1] International Telecommunication Union (ITU): "Open Systems Interconnection – Base Reference Model", ITU-T, Recommentation X.200, July 1994.

[2] R. Braden, T. Faber, M. Handley: "From protocol stack to protocol heap: role-based architecture", SIGCOMM Computer Communication Review, vol. 33, no. 1, pp. 17–22, 2003.

[3] C. Henke, A. Siddiqui, R. Khondoker, "Network Functional Composition: State of the Art", In Proceedings of the IEEE ATNAC, Nov. 2010.

[4] M. Becke, T. Dreibholz, H. Adhari, E. P. Rathgeb: „On the Fairness of Transport Protocols in a Multi-Path Environment", in Proceedings of the IEEE International Conference on Communications (ICC), Ottawa, Canada, June 2012.

[5] C. Raiciu, M. Handley, D. Wischik: „Practical Congestion Control for Multipath Transport Protocols", Technical Report, University College London, London, UK, 2009.

[6] I. Psaras, L. Wood, R. Tafazolli: "Delay-/Disruption-Tolerant Networking: State of the Art and Future Challenges", Technical Report, University of Surrey, UK, 2010.

[7] Web page: http://www.homer-conferencing.com.

[8] Web page: http://www.ekiga.org.

[9] Web page: http://www.linphone.org.

[10] J. Rosenberg, H. Schulzrinne et. Al: "SIP: Session Initiation Protocol", IETF RFC 3261, June 2002.

[11] M. Handley, V. Jacobson: "SDP: Session Description Protocol", IETF RFC 3266, April 1998.

[12] Web page: http://sofia-sip.sourceforge.net.

[13] G. Mulligan, D. Gracanin: "A comparison of SOAP and REST implementations of a service based interaction independence middleware framework", in Proceedings of Winter Simulation Conference (WSC), pp.1423-1432, Austin, Texas – USA, Dec. 2009.

[14] Internation Telecommunication Union (ITU): "Infrastructure of audiovisual services – Systems and terminal equipment for audiovisual services", ITU-T, Recommendation H.323, Dec. 2009.

[15] F. Liers et. Al: "GAPI: A G-Lab Application-to-Network Interface", 11th Workshop on IP: "Visions of Future Generation Networks" (EuroView2011) , Würzburg, Germany, Aug. 2011.

[16] Web page: http://www.german-lab.de.

[17] F. Liers, T. Volkert, A. Mitschele-Thiel: "The Forwarding on Gates Architecture: Merging IntServ and DiffServ", International Conference on Advances in Future Internet (AFIN) 2012 , Rome, Italy, Aug. 2012.

[18] Web page: http://www.ffmpeg.org.

[19] H. Schulzrinne et. Al: "RTP: A Transport Protocol for Real-Time Applications, IETF RFC 3550, July 2003.

[20] L. A. Larzon et. Al: "The Lightweight User Datagram Protocol (UDP-Lite)", IETF RFC 3828, July 2004.

[21] R. Stewart: "Stream Control Transmission Protocol", IETF RFC 4960, Sept. 2007.

[22] T. Volkert, F. Liers: "Video transcoding and rerouting in Forwarding on Gates networks", 12th Workshop on IP: "Visions of Future Generation Networks" (EuroView2012) , Würzburg, Germany, Aug. 2012.

[23] T. Volkert, A.Mitschele-Thiel: "Hierarchical routing management for improving multimedia transmissions and QoE", in Proceedings of IEEE WoWMoM, San Francisco, June 2012.

[24] M. Becke, T. Dreibholz, H. Adhari, E. P. Rathgeb: "A Future Internet Architecture supporting Multipath Comm. Netw.", in Proceedings of the IEEE NOMS, Maui, Hawaii/U.S.A, 2012.

[25] T. Volkert, F. Liers, M. Becke, H. Adhari: "Requirements-oriented Path Selection for Multipath Transmission", 12th Workshop on IP: "Visions of Future Generation Networks" (EuroView2012), Würzburg, Germany, Aug. 2012.

[26] Web page: http://www.bigbuckbunny.org/index.php/download/.

[27] S. Deering, R. Hinden: "Internet Protocol, Version 6 (IPv6) Specification", IETF RFC 2460, December 1998.

[28] F. Evers, P. Drieß, M. Brückner: "Demonstrating MoSaKa - A QoS System for Unstable Links With High Delay", 12th Workshop on IP: "Visions of Future Generation Networks" (EuroView2012) , Würzburg, Germany, July 2012.