Kvazzup: Open Software for HEVC Video Calls

Joni Räsänen, Marko Viitanen, Jarno Vanne, Timo D. Hämäläinen
Laboratory of Pervasive Computing
Tampere University of Technology, Finland
{joni.rasanen, marko.viitanen, jarno.vanne, timo.d.hamalainen}@tut.fi

***Abstract*— This paper introduces an open-source HEVC video call application called Kvazzup. This academic proposal is the first HEVC-based end-to-end video call system with a user-friendly Graphical User Interface for call management. Kvazzup is built on the Qt framework and it makes use of four open-source tools: Kvazaar for HEVC encoding, OpenHEVC for HEVC decoding, Opus codec for audio coding, and Live555 for managing RTP/RTCP traffic. In our experiments, Kvazzup is prototyped with low-complexity VGA and high-quality 720p video calls between two desktops. On an Intel 4-core i5 processor, the VGA call accounts for 17% of the total CPU time. Averagely, it requires a bit rate of 0.31 Mbit/s out of which 0.26 Mbit/s is taken by video and 0.05 Mbit/s by audio. In the 720p call, the respective figures are 46%, 1.13 Mbit/s, 1.08 Mbit/s, and 0.05 Mbit/s. These test cases also validate the feasibility of HEVC in different types of video calls. HEVC coding is shown to account for around 34% of the Kvazzup processing time in the VGA call and 45% in the 720p call.**

Keywords—Video call; High Efficiency Video Coding (HEVC); Real-time Transport Protocol (RTP); peer-to-peer; Kvazzup video call application

I. Introduction

Globally, IP video traffic is forecast to grow threefold from 2016 to 2021 [1]. One of the drivers behind this growth is video communications fostered by advanced video features in consumer devices, faster IP networks, and popular Internet video telephony services such as Skype [2] with 300 million active users already. Particularly, two-party video calls and multi-party video conferencing are increasingly used in the business sector where IP traffic is expected to be three times as high by 2021 [1]. Mitigating this exponential growth means taking efficient video compression into use in video call applications.

The latest international video coding standard, *High Efficiency Video Coding* (*HEVC/H.265*) [3] reduces bit rate by almost 40% over the preceding state-of-the-art standard AVC/H.264 [4] for the same objective visual quality but at around 40% encoding complexity overhead [5]. Thus, HEVC is a promising candidate for video telephony applications to deliver high-quality video content at low a bit rate.

Up to this date, many video telephony systems have been released but most of them are commercial products whose features and operating principles are confidential. Therefore, this work only focuses on open-source solutions out of which the most notable ones are Linphone [6], Ekiga [7], BareSIP [8], Empathy [9], KPhone [10], Ring [11], Tox [12], yate [13], and Jitsi [14]. Among these solutions, BareSIP is the only one that supports HEVC, but it is only a command-line software without a *Graphical User Interface* (*GUI*).

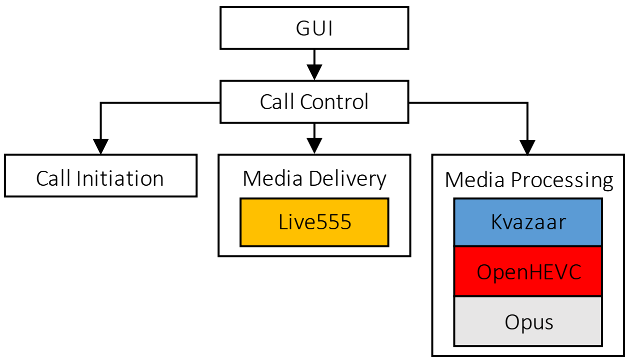

Figure 1. High-level architecture of Kvazzup.

Kvazzup is another HEVC-based video telephony system initiated and developed by our Ultra Video Group [15]. The main motivation behind this academic open-source project is to provide an end-to-end system with the latest video and audio codecs for high-quality video calls at low bandwidth. In addition, we seek to increase user friendliness with a clear GUI. Kvazzup version 0.2 was used in this work and the source code for it can be found on GitHub [16]. It is licensed under the ISC license [17].

Kvazzup is written in C++ and built on Qt v5.9 framework [18], which provides many features such as camera input, audio input/output, and threading. Kvazzup also makes use of the following open-source tools: Kvazaar v1.1 [19] for HEVC encoding, OpenHEVC v2.0 [20] for HEVC decoding, Opus Codec v1.21 [21] for audio encoding and decoding, and Live555 Streaming Media v0.90 [22] for RTP/RTCP traffic management. Currently, Kvazzup operates on Windows and it can be compiled using MinGW.

The remainder of this paper is organized as follows. Section 2 presents the overall architecture of Kvazzup including its main components and multimedia tools. Section 3 gives results of experimental measurements. Section 4 concludes the paper.

II. Kvazzup Video Call System

Fig. 1 depicts the overall architecture of Kvazzup. It can be divided into five main components: 1) GUI; 2) Call Control; 3) Call Initiation; 4) Media Delivery; and 5) Media Processing.

A. Graphical User Interface (GUI)

Fig. 2 presents a snapshot of Kvazzup call between two

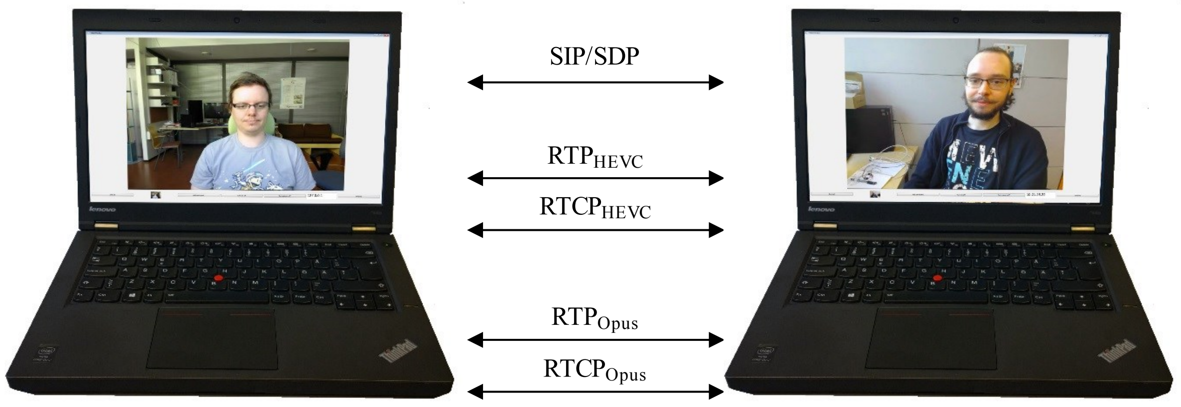

Figure 2. A two-way video call in Kvazzup.

participants. A GUI of a Kvazzup client can be seen on the laptop screens. The GUI provides the user with the following features: 1) starting a call to an IP address, 2) turning on/off the microphone or the camera, 3) displaying a statistics window; and 4) ending a call. The GUI includes a video widget of the other call participant and a mirror-like preview for the user to check the orientation of the camera. A pop-up appears when someone attempts to call the user and the user can accept or decline the call.

B. Call Control

Call control is responsible for managing the interactions between the user and different components of the application. Upon the user's request, the Call Control instructs Call Initiation to send a message to the contact of interest. If the recipient accepts the call, Call Control is responsible for linking Media Delivery and Media Processing together and terminating them when the call ends.

C. Call Initiation

Call Initiation is responsible for *Session Initiation Protocol* (*SIP*) communication. SIP [23] is an RFC standard that describes a protocol for facilitating communication regarding the start and termination of a video call. SIP is paired with *Session Description Protocol* (*SDP*) [24] that describes video call parameters such as ports and media types used.

The SIP implementation includes starting and ending a video call using SIP requests and responses. Fig. 3 shows how these messages and responses are processed within Call Initiation. Call Negotiation keeps track of existing sessions and their state. The messages are composed to a string using the SIP Composer. Connection module maintains the *Transmission Control Protocol (TCP)* connection via a socket. Outgoing SIP messages are sent and incoming SIP messages are read using this socket. In the receiving end, the SIP Parser parses incoming SIP messages and the Call Negotiation checks their validity.

D. Media Delivery

Live555 is responsible for media delivery via *Real-time Transport Protocol* (*RTP*) and delivery quality monitoring via

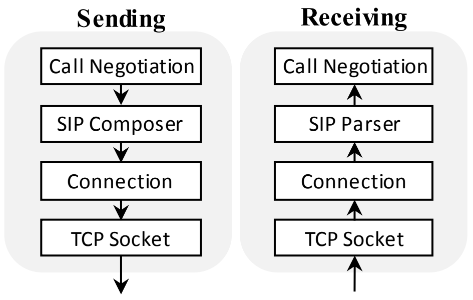

Figure 3. Call Initiation flow.

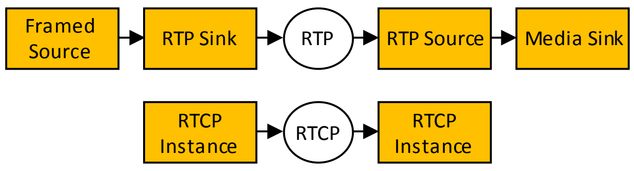

Figure 4. RTP/RTCP delivery flow with Live555.

RTP Control Protocol (*RTCP*). The RTP and RTCP protocols are specified in detail in [25]. Kvazzup sends and receives separate RTP and RTCP streams for video and audio.

Fig. 4 presents how Kvazzup uses Live555 to stream RTP and RTCP packets across a network. Framed Source gets its input from the sending filter graph and Media Sink forwards the received media to the receiving filter graph. There is one RTCP Instance associated with both sending and receiving. Live555 applies RTP sinks to receive data and RTP sources to send data. It operates in a separate thread and uses callbacks to communicate when ready to send or receive data.

E. Media Processing

Kvazzup performs media processing with a filter graph in which a series of filters are connected to each other in a serialized way and each filter has an individual task. In video filter graph, these tasks are: 1) generating samples; 2) changing the format of samples; 3) encoding samples; 4) transporting samples across a network; 5) decoding samples; and 6) presenting them to the user. Audio processing also includes all these tasks expect for the format conversion.

The Qt main thread handles the processing of input and output devices excluding the camera. The Live555 thread processes the Live555 filters and the rest of the filters have a Qt thread which continuously processes data. The filters in Kvazzup have a list of output filters that define where a generated or processed sample is sent for further processing. When the filters do not have anything to process, they sleep. The incoming sample is stored into an input buffer and the filter wakes up to process samples from the buffer until the buffer is empty.

Fig. 5 and Fig. 6 describe the video and audio filter graphs of Kvazzup, respectively. Each box in these graphs is a filter and each arrow shows how the data moves. A data flow for sending is shown on the left and a data flow graph for receiving on the right. Both the video and audio use a Framed Source filter for sending media over a network and a Media Sink filter for receiving it.

The video graph starts at the Camera filter that captures frames from a web camera using the Qt camera module. The module outputs video in RGB32 format which is converted to YUV420 format for Kvazaar. The Kvazaar filter encodes the frames to HEVC video and Live555 sends them across the network to OpenHEVC for decoding frames back to YUV420 format. Since Qt is not able to show YUV420 coded frames, they have to be converted to RGB32 format for display. The frame is upside down when it arrives at the Display filter, which needs to mirror the image to the correct orientation before passing it to a video widget for drawing the frame on a screen. The output from the Camera filter is also attached to a second Display filter to show the user's own video feed.

The audio filter graph starts from the Audio Capture filter, which uses Qt for audio input. The audio format is *Pulse Code Modulation (PCM)* single channel audio with a sample rate of 48000 Hz. The Opus encoding and decoding is carried out by the reference codec [21] and streaming is done using Live555. The Audio Output filter plays the received sound samples.

III. Experimental Results

Kvazzup was benchmarked by running a two-party video call on two Windows 7 desktops powered by Intel i5-4570 processor and 16 GB of RAM. The same parameters were set in Kvazzup clients at both ends. The video streams were shot by Logitech c930s cameras in a well-illuminated room. Both streams contained a single person with limited movement.

The complexity of video encoding is a large factor in video call applications, and therefore the encoding parameters in Kvazaar are relevant to the performance analysis. The Kvazzup clients applied Kvazaar ultrafast preset for HEVC video encoding. Kvazaar was parametrized to use wavefront parallel processing [3] with four threads. One keyframe was inserted every 64 frames and each keyframe included parameter sets. The statistics window of Kvazzup remained hidden and the viewed video was not scaled.

Performance evaluations were conducted with two call configurations of Kvazzup: 1) a low-complexity setup with VGA resolution (640 × 480) and a *quantization parameter* (*QP*) value of 32 in Kvazaar; 2) a high-quality setup with 720p (1280 × 720) resolution and a QP value of 27. The frame rate was 30 *frames per second* (*fps*) in both setups.

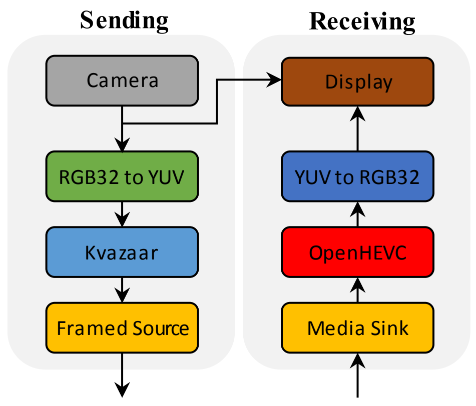

Figure 5. Video Filter Graph.

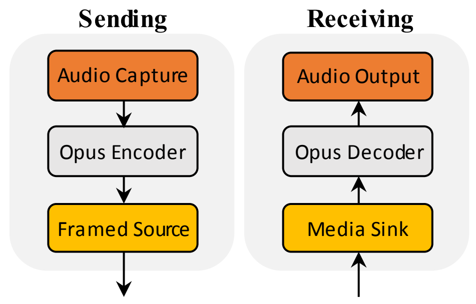

Figure 6. Audio Filter Graph.

The complexity analysis was done with Process Explorer version v16.20 0 by taking three snapshots of the running processes at 30 seconds intervals and then averaging them. On average, the VGA call used 17.3% and 720p call 45.6% of total available CPU time. Hence, Kvazzup can be run with 720p resolution at 30 fps on a modest desktop without performance limitations.

Fig. 7 and Fig. 8 present the CPU usage breakdown between the most compute-intensive parts of Kvazzup in VGA and 720p calls, respectively. The most notable responsibilities for the "Qt main" include audio input/output and drawing the images on screen. The "Other" category includes system libraries used by Qt. The "Camera" category represents a Windows system library which Qt uses to capture images. Live555 is responsible for media delivery in its entirety along with RTCP. The rest of the categories are filters in Media Processing. Opus encoding and decoding have been combined under "Opus" and both display filters are under "Displays".

Camera, Kvazaar, OpenHEVC, format conversions, and Display take 69% of the Kvazzup processing time in VGA call and 84% in 720p call. If we assume that the rest of the processing time is evenly distributed between audio and video, the shares of the video and audio are 84 % and 16% in the VGA call. In 720p call, the respective percentages are 92% and 8%.

The obtained results also validate the feasibility of HEVC in video call applications. HEVC coding (Kvazaar + OpenHEVC) is shown to account not more than 33.9% of the Kvazzup processing time in the VGA call and 45.3% in the 720p call.

The bit rate analysis was done with Windows Resource Monitor. Average video bit rates for the VGA and 720p calls were 0.31 Mbit/s and 1.13 Mbit/s, respectively. The audio bit rate was constant 50 kbit/s in both cases.

IV. Conclusions

This paper presented an end-to-end HEVC video call system Kvazzup. It is the first open-source solution featuring the latest video and audio codecs for economic communication as well as a GUI for convenient call management. The proposed setup uses Kvazaar for HEVC encoding, OpenHEVC for HEVC decoding, Opus codec for audio coding, Live555 for media delivery, and SIP for iniating and ending a call. The system also validates the feasibility and benefits of HEVC in a video call. Kvazzup is capable of running on a modest desktop at 720p30 resolution.

In the future, the proposed system will be extended to support multi-party video conferencing and higher video resolutions. This way, communication between larger number of participants can be served with improved user experience.

Acknowledgment

This work was supported in part by the European Celtic-Plus Project Virtuose and the Academy of Finland (decision no. 301820).

References

[1] Cisco, *Cisco Visual Networking Index: Forecast and Methodology*, 2016-2021, Jun. 2017.

[2] *Skype* [Online]. Available: http://www.skype.com/

[3] *High Efficiency Video Coding*, document ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T and ISO/IEC, Apr. 2013.

[4] *Advanced Video Coding for Generic Audiovisual Services*, document ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), ITU-T and ISO/IEC, Mar. 2009.

[5] J. Vanne, M. Viitanen, T. D. Hämäläinen, and A. Hallapuro, "Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1885-1898.

[6] *Linphone open-source voip software* [Online]. Available: http://www.linphone.org/

[7] *Ekiga* [Online]. Available: http://www.ekiga.org/

[8] *BareSIP* [Online]. Available: http://creytiv.com/baresip.html

[9] *Empathy* [Online]. Available: https://wiki.gnome.org/Apps/Empathy

[10] *KPhone* [Online]. Available: https://sourceforge.net/projects/kphone/

[11] *Ring* [Online]. Available: https://ring.cx/

[12] *Tox* [Online]. Available: https://tox.chat/

[13] *Yet* Another Telephony Engine [Online]. Available: http://www.yate.ro/

[14] *Jitsi* [Online]. Available: https://jitsi.org/

[15] *Ultra video group* [Online]. Available: http://ultravideo.cs.tut.fi/

[16] *Kvazzup* [Online]. Available: https://github.com/ultravideo/kvazzup

[17] ISC License [Online]. Available: https://opensource.org/licenses/ISC

[18] *Qt* [Online]. Available: https://www.qt.io/

[19] *Kvazaar HEVC encoder* [Online]. Available: https://github.com/ultravideo/kvazaar

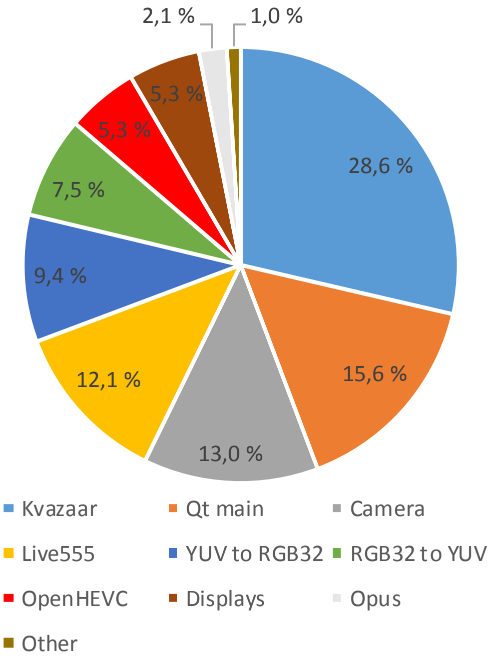

Figure 7. Relative CPU usage with VGA resolution and QP 32.

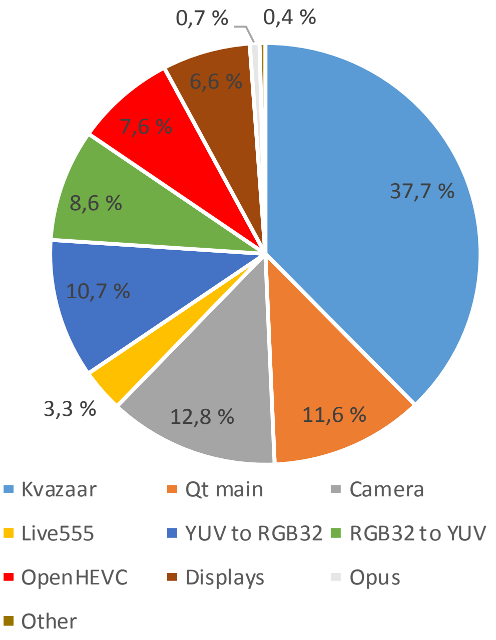

Figure 8. Relative CPU usage with 720p resolution and QP 27.

[20] *OpenHEVC* [Online]. Available: https://github.com/OpenHEVC/openHEVC

[21] *Opus Codec* [Online]. Available: http://opus-codec.org/

[22] *LIVE555* [Online]. Available: http://www.live555.com/

[23] *IETF RFC 3261 SIP: Session Initiation Protocol* [Online]. Available: https://www.ietf.org/rfc/rfc3261.txt

[24] *IETF RFC 4566 SDP: Session Description Protocol* [Online]. Available: https://www.ietf.org/rfc/rfc4566.txt

[25] *IETF RFC 3550 RTP: A Transport Protocol for Real-Time Applications* [Online]. Available: https://www.ietf.org/rfc/rfc3550.txt

Process Explorer, Microsoft [Online]. Available: https://technet.microsoft.com/en-us/sysinternals/processexplorer.aspx