J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

An Advanced Coding for Video Streaming System: Hardware and Software Video Coding

Tuan Thanh Le[1] and Eun-Seok Ryu[2]
[e-mail: tuanlt@gc.gachon.ac.kr, esryu@skku.edu]

Abstract

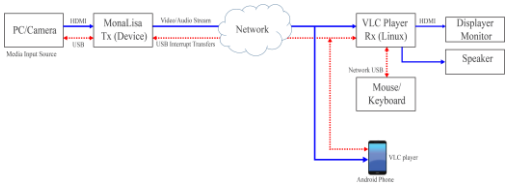
Currently, High-efficient video coding (HEVC) has become the most promising video coding technology. However, the implementation of HEVC in video streaming systems is restricted by factors such as cost, design complexity, and compatibility with existing systems. While HEVC is considering deploying to various systems with different reached methods, H264/AVC can be one of the best choices for current video streaming systems. This paper presents an adaptive method for manipulating video streams using video coding on an integrated circuit (IC) designed with a private network processor. The proposed system allows to transfer multimedia data from cameras or other video sources to client. For this work, a series of video or audio packages from the video source are forwarded to the designed IC via HDMI cable, called Tx transmitter. The Tx processes input data into a real-time stream using its own protocol according to the Real-Time Transmission Protocol for both video and audio, then Tx transmits output packages to the video client though internet. The client includes hardware or software video/audio decoders to decode the received packages. Tx uses H264/AVC or HEVC video coding to encode video data, and its audio coding is PCM format. By handling the message exchanges between Tx and the client, the transmitted session can be set up quickly. Output results show that transmission's throughput can be achieved about 50 Mbps with approximately 80 msec latency.

☞ keyword : Video streaming, RTP, H.264, HEVC, video coding

[1] Dept. of Computer Engineering, Gachon University, Seongnam, 13342, South Korea
[2] Dept. of Computer Education, Sungkyunkwan University, Seoul, 03063, South Korea
*Corresponding author: Eun-Seok Ryu
[Received 25 May 2020, Reviewed 30 May 2020, Accepted 30 May 2020]
☆ Acknowledgement : This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2019-2017-0-01630) supervised by the IITP(Institute for Information & communications Technology Promotion).

1. Introduction

Recently, the HEVC [1] has become the top technology in the field of video coding. Compared to the predecessor video coding H264, HEVC (H.265) is an efficient technology that allows saving twice bandwidth while unchanged video resolution. The first version of HEVC achieved an approximately 50% bitrate reduction compared to its predecessor H.264/AVC with equivalent subjective quality [2]. Over the past few years, many video transmission systems, CCTV systems, or online video services have gradually moved to the HEVC application, which increases bandwidth responsiveness. Hence, current video service platforms need to upgrade their construction to support HEVC coding. However, HEVC implementation also has limitations such as high cost, system complexity, compatibility with existing video service systems. Therefore, in current times, the H.264-AVC codec [3] is still an effective choice. Video codec H.264/AVC is able to encode and decode video at a low cost in terms of processing time and resource usage. The main drawback of H.264 is bandwidth when compared to H.265/HEVC. Especially in real-time streaming systems, the bandwidth limitation has become even more urgent in multi-channel contexts. Hence, to advance the efficiency of H.264 video streaming services, we propose a solution to improve the streaming's bandwidth using a private designed IC that comes with the H.264 video codec and PCM audio inside. We implemented the designed IC within a private network processor to efficiently embed the H.264/AVC codec. By deploying the encoder and decoder under kernel space in both sides (server and client), the proposed system is able to minimize processing time thereby improving overall performance.



(Figure 1) The conceptual architecture of the proposed system

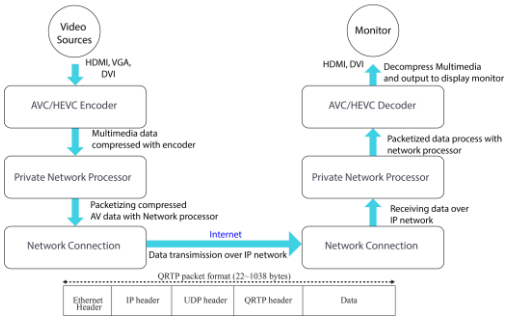
In addition, to increase the performance of video streaming over the internet environment, we also developed an adaptive transport protocol based on the Real-Time Transport Protocol (RTP). The implementation of the customized protocol has been done according to [4] and [5]. The customized protocol called QRTP, allows the video streaming system to be able to meet real-time stream with an average bitrate of approximately 50 Mbps. Finally, the H.264 and QRTP are deployed under Linux kernel space, then this idea allows us to apply HEVC codec into designed IC similar to H.264.

The remainder of this paper is divided into sections as follows: Section 2 describes the main idea of the proposed method based on the video coding point of view and RTP protocol. Section 3 provides the outcome of the performance of the proposed system. Finally, section 4 shows the conclusions of the proposed method and gives further research.

2. QRTP Streaming System

2.1 The proposed streaming system

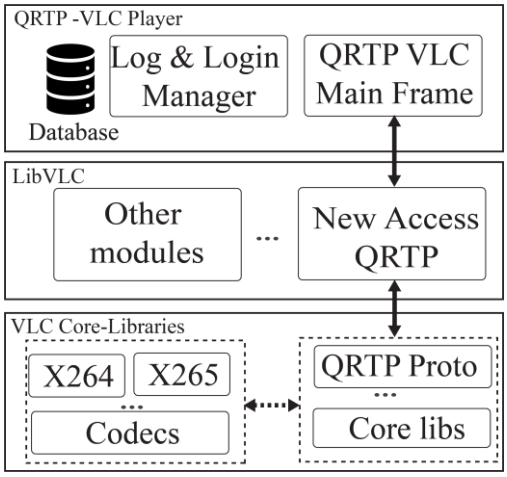
As shown in Figure 1, the proposed solution provides an overview of efficient streaming service according to video encoder and decoder on designed IC. Video sources such as cameras, mini PC, etc. transmit video frames into the Tx device via HDMI connection. Next step, Tx's encoder encodes these frames into a video bitstream using H.264 or HEVC codecs. The video bitstream is organized as a sequence of Network Abstract Layer (NAL) packets. To transmit NAL packets to client, QRTP will attach each NAL packet as the payload field of a QRTP packet. We implemented a real-time transport protocol QRTP based on RTP [5].



(Figure 2) Functional flow of system with Tx/Rx.

The proposed system includes two kinds of client: hardware designed IC (Rx) and software VLC video/audio playback [6]. The designs of Rx are almost like Tx except for its decoder. To handle video processing tasks, Tx and Rx use a private network processor, which was embedded on our integrated circuit. As shown in Figure 2, the packet flow of the proposed system is described in case using Tx and Rx. Moreover, all tasks are deployed under kernel space to reduce processing time. In this

way, the proposed system can provide a high-bandwidth connection for video streaming. If the video client uses a VLC player as a video client, the VLC player can handshake pairing with Tx as the same as Rx. The QRTP protocol over UDP/IP stack includes QRTP packet format for each layer of internet model. Therefore, H.264 or H.265 NAL packets are encapsulated as payload (data part) of QRTP packet. The length of QRTP packet is around 22 bytes ~ 1038 bytes for audio, video, and USB interrupt transfer data. The ethernet header length is 14 bytes, IP header length is 20 bytes, and UDP header length is 8 bytes.

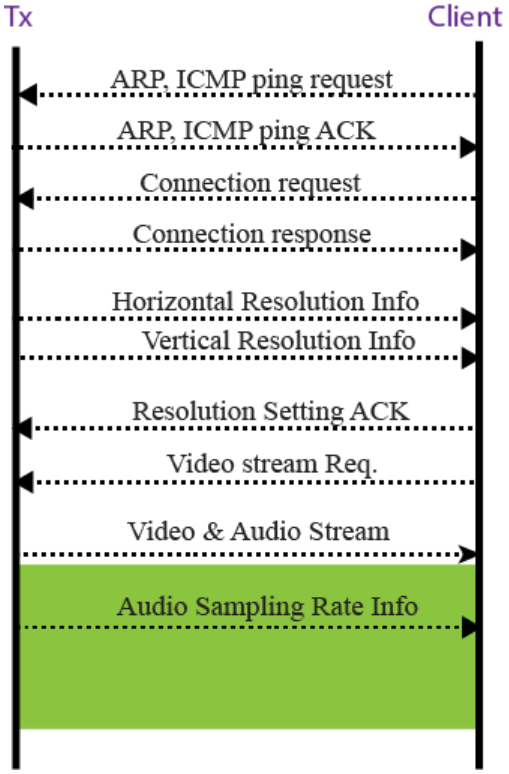


(Figure 3) QRTP-VLC Structure

(Table 1) Payload Type Map of QRTP

Payload	Description	Direction
0x71	Connection request	Rx->Tx
0x72	Connection response	Tx->Rx
0x78	Video resolution: vertical size	Tx->Rx
0x79	Video resolution: horizontal size	Tx->Rx
0x7A	Resolution ACK	Rx->Tx
0x7F	Video stream req.	Rx->Tx
0x00	Video stream data	Tx->Rx
0x03	Audio stream data	Tx->Rx
0x7C	Audio sampling rate	Tx->Rx
0x7B	Reset flow	Tx->Rx

Figure 3 shows that the VLC video playback implementation with QRTP protocol access module inside [7]. Furthermore, the modified VLC version also allows configurable options for QRTP as same as RTP. QRTP protocol was implemented based on the original RTP. As shown in table 1, the QRTP payload type (PT) map was exactly designed to support both procedures: pairing flow and video/audio streaming procedure. PT value in table 1 is one byte in hexadecimal format. For example, the pairing flow exchanges control messages between Tx and client. This procedure will send/receive packets with payload types such as 0x71, 0x72, 0x78 0x79, 0x7A, and 0x7F. Those others are for video streaming session or session control.



(Figure 4) Pairing flow between Tx and VLC/ Rx client

The pairing flow procedure allows client to send requests to Tx and establishing a connect session between them. Then, pairing flow will initialize parameters for video/audio sessions before stream coming to client. Figure 4 shows that the pairing flow can be divided step-by-step as follows:

1) MAC ARP and ICMP pings: To clearly trace the internet route between client and server. Additionally, these pings also allow Tx server and client to correctly verifying their partner.

2) Connection request/reply: Establishing a connection and making ACK message exchanges.

3) Resolution confirmation: Allows Tx to send resolution in detail double times for setting vertical size and horizontal size.

4) Resolution ACK: Client replies ACK to Tx to confirm that resolution setting was done.

5) Video stream request: Everything is ready. Client sends a video stream request to Tx.

6) Video & audio data: Tx streams video and audio data to client.

7) Audio sampling rate: while Tx was streaming video/audio, it can send audio sampling rate in detail to client. The client does confirmation and reconfigure its audio parameters.

2.3 Audio streaming

For multimedia support, an audio stream (with PT 0x03) is also integrated into parallel with a video stream. As illustrated in Figure 1, the audio stream can be transmitted to the client on the audio port, which is determined by the video port plus 1. The audio data is composed of 4 bytes. Figure 5 below shows the audio data structure. "Left" data when the LR (Left-Right) bit is '0' and "Right" data when it is '0'. The audio data is composed of 3 bytes. Audio stream format is PCM-32 with support stereo sound.

Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved							LR	Data [23:16]							
	Data [15:0]															

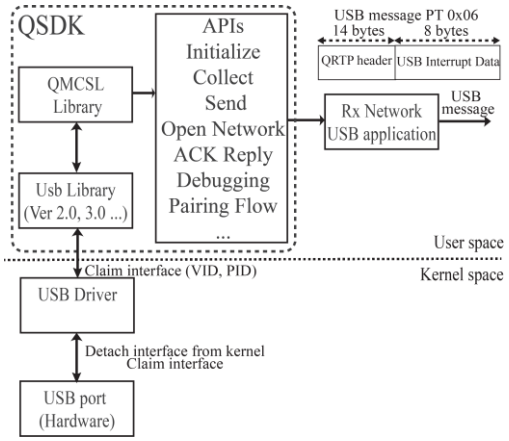
(Figure 5) Audio data packet structure

2.3 Network USB

To enable user-client can control video source, the proposed method uses network USB on video client Rx to support I/O devices such as mouse and keyboard. As shown in Figure 1, network USB allows Rx to send interrupt transfer signals of mouse/keyboard to Tx by using QRTP protocol. The payload size of interrupt transfer is 8 bytes. Tx gets these interrupt transfers and transform them into mouse/keyboard events at video source. Currently, network USB supports ports version 2.0 or 3.0. Specified details of interrupt transfers can be reviewed in [8], [9].

Network USB includes an SDK as in Figure 6, which's called QSDK. The main core of SDK was implemented as a private library (QMCSL lib) based on libusb-1.0. QSDK provides APIs that allows developing application regarding USB devices. For example, a Rx network USB application can use APIs to create a connect session to Tx, or send USB message of 22 bytes to Tx, etc.

The APIs of QSDK can be described as follows: Initialize function: QSDK initializes USB interface list, check parameters according to vendor ID (VID) and product ID (PID); Opening sessio function: QSDK detach specified interface from kernel space to take the authorship; Claiming interface function: QSDK claim interface based on VID, PID to create new session; Open Network function: Open UDP socket to connect to Tx device via UDP port; Gathering transfers function: QSDK interacts with USB device to collect interrupt transfers from USB device by exchanging messages; Send function: Allows application sending USB interrupt transfer to Tx device; Closing session function: QSDK send a message to close all transfers. Release authorship, clear memory and remove temporary storage. And some other functional APIs for utility and statistic.



(Figure 6) QSDK architecture and work flow

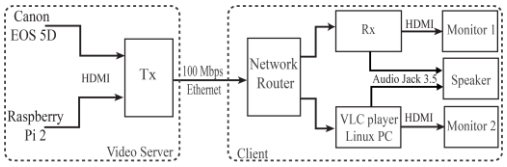
3. Performance Evaluation

In order to prove the real performance of the proposed system, we set up a testbed as shown in Figure 7. A powerful PC was set up with a Core i7-7700K 4.2 GHz processor, 32 GB of memory with Linux Ubuntu OS 18.04 GCC 7.4.0 as VLC client. We set up a Canon EOS 5D camera [10] an input video camcorder. A Raspberry Pi 2 Model B [11] also was installed as the video source. Additionally, we used two designed ICs board Tx and Rx as server and client, respectively. The diagram in Figure 8 shows the working flow of Tx encoder and Rx decoder in detail.

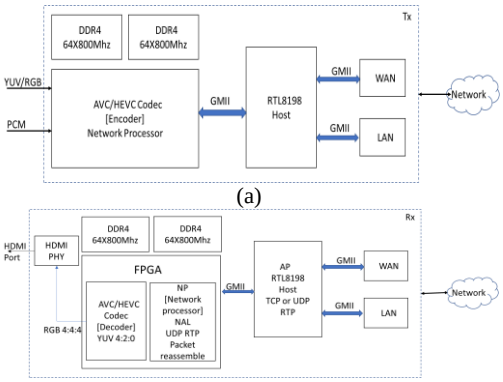
Figure 9 shows the average throughput of video stream around 49.57 Mbps for video stream with 1920x1080 (1080p) resolution, and 19.46 Mbps with 1080x720 (720p) resolution. Additionally, the packet loss rate not over 0.45 % in any case. Despite, QRTP is based on RTP over UDP/IP stack, the packet loss rate of 0.43% allows the proposed system can handle the real-time stream. Furthermore, the proposed system can apply the Forward Error Correction (FEC) [12] to recover the packet loss issue. This leads to QRTP with FEC can decrease the packet loss rate to less than 0.1%.

We used the FFmpeg library with dependent libraries [13] as evaluation software. Additionally, the VLC software version 4.0.0-dev was used as video playback of the video client. H.264 video encoder and decoder support resolution from 16x16 (minimum) to 1920x1080 (maximum). Video codec H.264 provides support for base, main and high profile within level 4.2 [14].

As shown in Table 2, the video stream with 1080p resolution always gets the latency that less than 82 msec. In the case of lower resolution, the latency is around 73~82 msec. The average latency for all possible resolutions is approximately 80 msec.



(Figure 7) The testbed's scenario



(Figure 8) (a) Tx encoder block diagram and (b) Rx decoder block diagram.

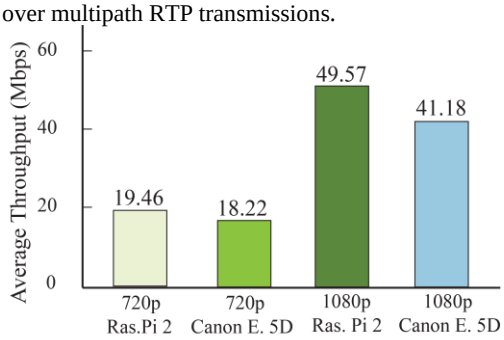
To analyze the quality of the decoder, according to [15] and [16], we progressed the comparisons between the original video at video sources and reconstructed video at video client. Table 3 shows that the performing "unsatisfactory" factor compared to other quality metrics when it comes to estimating the quality of images and videos as perceived by humans according to Y channel PSNR values. We confirmed the comparisons for both 1080p and 720p videos. Output results proved that all Y-PSNR values are higher than 38 (dB) threshold. This means that the quality of reconstructed videos at the client is reasonable to feel fully immersed in video streaming as perceived by humans.

(Table 2) The delay for various video sources

Video Source	720p	1080p
Raspberry Pi 2	71.36 msec	78.93 msec
Canon EOS	73.47 msec	81.52 msec

The audio stream is stored as a .pcm file in Rx client in detail (Audio format :pcm; Sampling rate: 12~192 KHz; channels: 2). This PCM file can be converted to AAC format before playing by a speaker. The audio sampling rate are vary from 12 kHz to 192 kHz. The most popular rates are 32 kHz, 44.1kHz and 48 kHz.

To verify the performance of the proposed method, we collected all experimental results and compared them to RTMP/H264 solution [17] and MDC adaptation [18]. The RTMP/H264 is a real-time streaming framework based on Realtime Messaging Protocol and H.264 encoding. MDC adaptation is network-adaptive multiple description coding (MDC) method for enhanced H.264 video streaming over multipath RTP transmissions.



(Figure 9) Average Throughput of QRTP/H.264 streaming

(Table 3) Y-PSNR for various video sources

Video source	720p	1080p
Raspberry Pi 2	41.46 dB	40.19 dB
Canon EOS 5D	38.25 dB	38.03 dB

As shown in table 4, our proposed method can provide the average throughput approximate 49.02 Mbps, while RTMP/H264 was giving 0.55 Mbps for wired-network communications and 54 Mbps for wireless communication. Moreover, the average delay of QRTP/H.264 is 80.04 msec far lower than RTMP/H264's delay of 141 msec. The packet loss rate (PLR) of the proposed method is around 0~0.43% without FEC that higher than RTMP/H264's PLR. The main reason is that RTMP/H264 was implemented based on TCP, and our QRTP/H.264 is based on UDP. By enabling FEC support [19], QRTP can give the PLR that lower than 0.1%. Additionally, from [20], we can confirm that the PLR value lower than 5% is proper for live video streaming service. Compared to the MDC adaptation solution, our proposed method also gives the PSNR value of 39.11 dB better than 38.85 dB at a packet loss rate lower than 5%. More details of the packet loss rate model can be reviewed in [17].

(Table 4) RTMP/H264 and QRTP/H.264 Comparison

Performance metrics	RTMP/H264	QRTP/H264
Average Throughput	0.55 Mbps 54 Mbps	49.02 Mbps
Average Packet Loss	0~0.4 %	0~0.43 % non-FEC
Average Delay	141.75 msec	80.04 msec

4. Conclusion

This paper presents an adaptive video streaming method by using efficient video codecs on both

hardware and software. The experimental results proved that our method can provide high-speed video streaming in real-time. The proposed system can provide video/audio streaming with a throughput of approximately 50 Mbps and latency around 80 msec. Furthermore, the proposed method also allows improving the video streaming systems such as upgrading HEVC video codec, extending service by adding control messages, and more important tasks. In the future, we are going to build a standard SDK that allows releasing packaged software.

References

[1] JCT-VC, "High Efficiency Video Coding". Available: https://hevc.hhi.fraunhofer.de/

[2] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video 344 coding standards—Including High-Efficiency Video Coding (HEVC)", IEEE Trans. Circuits Syst. Video Technol., 345 vol. 22, no. 12, pp. 1669–1684, Dec. 2012.

[3] ST. Wiegand and Gary J. Sullivan, "Overview of the H.264/AVC Video Coding Standard", IEEE Transactions on 349 Circuits and Systems for Video Technology, Vol.13, No.7, p.p 560 - 576, IEEE, July.2003.

[4] IETF, RFC 1889, "Real-Time Transport Protocol RFC 1889". Available: https://tools.ietf.org/html/rfc1889

[5] IETF, "RTP Profile for Audio and Video Conferences with Minimal Control RFC 1890".Available: https://tools.ietf.org/html/rfc1890

[6] VLC, "Video Lan Player". Available: https://www.videolan.org/vlc/index.html

[7] VLC, "VLC core library – modules". Available: https://wiki.videolan.org/Documentation:Modules/

[8] Changyi Gu, "Building Embedded Systems: Programmable Hardware", 2016.

[9] Beyond Logic, "USB in a Nutshell", 2020. Available: https://www.beyondlogic.org/usbnutshell/usb4.shtml#Interrupt

[10] "Raspberry Pi 2 model B product". Available. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[11] Canon, "Canon EOS 5D Mark IV". [Online]. Available: https://www.usa.canon.com/internet/portal/us/home/products/details/cameras/eos-dslr-and-mirrorless-cameras/dslr/eos-5d-mark-iv

[12] G. Charlet and P. Pecci, "Undersea Fiber Communication Systems (The Second Edition) – Forward Error Correction", 2016.Available: https://www.sciencedirect.com/topics/engineering/forward-error-correction

[13] FFMPEG, "FFmpeg software version 4.0.2 and document guide". [online]. Available: https://ffmpeg.org/download.html#get-sources

[14] MPEG-4 AVC/H.264 High Profile / Level 4.2 Video Compression. [online]. Available: http://dicom.nema.org/medical/dicom/2017e/output/chtml/part05/sect_A.4.7.html

[15] Imran Ullah Khan et al., "Performance Analysis of H.264 Video Decoder: Algorithm and Applications", 2015 International Conference on Energy Economics and Environment (ICEEE), Mar. 2015.

[16] S. Paulikas, "Estimation of Video Quality of H.264/AVC Video Streaming", EuroCon 2013, July 2013.

[17] A. Nurrohman and M. Abdurohman, "High Performance Streaming Based on H264 and Real Time Messaging Protocol (RTMP)", 2018 6th International Conference on Information and Communication Technology (ICoICT), May 2018.

[18] P. Correia, P. Assuncao and V. Silva, "Enhanced H.264/AVC Video Streaming using Network-adaptive Multiple Description Coding", 2011 IEEE EUROCON, April 2011.

[19] VLC with FEC support. Available: https://github.com/n2i911/vlc-with-fec

[20] Cisco Video Quality of Service. [online]. Available: https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-video/212134-Video-Quality-of-Service-QOS-Tutorial.html