Paper

Video Streaming Framework

Andrzej Buchowicz and Grzegorz Galiński

Institute of Radioelectronics, Warsaw University of Technology, Warsaw, Poland

Abstract—The framework for testing video streaming techniques is presented in this paper. Short review of error resilience and concealments tools available for the H.264/AVC standard is given. The video streaming protocols and the H.264 payload format are also described. The experimental results obtained with the framework are presented in this paper too.

Keywords—error resilience and concealment, H.264/AVC, media delivery over IP networks, video coding.

1. Introduction

The video coding technology has been rapidly developing during the last years. Broadband networks have been growing even faster. Media delivery over the IP networks has been widely accepted and it seems it may replace the traditional media distribution methods in near future. However, the network performance depends on many factors and may not always guarantee the required quality of the transmitted media. It is extremely important in many application to increase the error resilience of the audio or video stream and to effectively conceal any errors that may occur during transmission.

The framework for testing video streaming techniques will be presented in this paper. It has been used as a tool for analysis and development of media adaptation, error resilience and error concealment algorithms. The framework has been limited to the streaming of the H.264/AVC encoded video with the use of RTP/RTCP protocol. However, it can be easily extended for other video codecs and transmission protocols. The experimental results obtained with this framework will also be presented.

1.1. H.264/AVC Bitstream

The international standard MPEG-4 H.264/AVC [1] is currently the most commonly used for video coding. Its first editions was released in 2003. The important enhancements: scalable video coding (SVC) and multiview video coding (MVC) were added in 2008 and 2009 respectively. The H.264/AVC standard is based on the hybrid motion compensation and transform algorithm [2]–[4] implemented in almost all preceding video coding standards, including MPEG-2 Video [5]. Many improvements of the classical algorithm significantly increased the H.264/AVC coding efficiency with respect to its predecessors. However, the coding efficiency was not the only objective for H.264/AVC standard developers. The video stream flexibility and its adaptability for different transmission channels was also an important factor. It has been achieved by separation of the signal processing from the transport-oriented processing – the H.264/AVC codec has been divided into two layers:

– video coding layer (VCL) – contains all compression tools, generates bitstream of the encoded macroblocks organized into slices;

– network abstraction layer (NAL) – encapsulates the bitstream generated by the VCL in units suitable for the transmission.

The H.264/AVC bitstream is a sequence of the NAL units (Fig. 1). Each NAL unit starts with one-byte header containing three fields:

– `F` – error indicator (1 bit), NAL unit with this field set to 1 should not be processed;

– `NRI` – NAL unit priority (2 bits), the value of this fields indicates the importance of the NAL unit for a video sequence reconstruction;

– `TYPE` – type of the NAL units (5 bits), values $0 \div 23$ are restricted to be used only within the H.264/AVC standard, values $24 \div 31$ may be used for other purposes, e.g., in transmission.

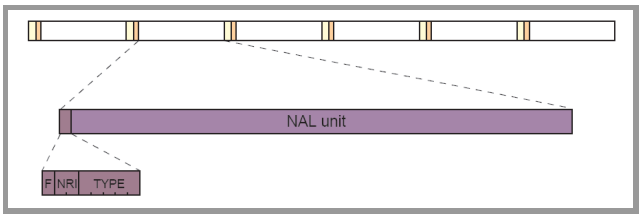

Fig. 1. H.264/AVC bitstream.

NAL units contain only data representing the encoded video sequence. Additional headers must be appended to each NAL unit to separate them. Annex B of the H.264/AVC standard defines such headers (start code – fixed byte sequence) for the transmission in byte-oriented networks (e.g., broadcasting). The headers used in packet-oriented networks will be discussed further in this section.

The H.264/AVC syntax is not as restricted as in its predecessors. There are no layers above the slice layer generated by the VCL. The higher level information are stored in the specific syntax elements: sequence parameter set (SPS) and picture parameter set (PPS). Special NAL unit type are assigned to carry the parameter sets. Several SPSs and PPSs can be defined and used by the encoder. Each macroblock in the H.264/AVC bitstream refers to the SPS and the PPS

which are used to encode it. Usually NAL units with parameter sets precede all other NAL units in the H.264/AVC bitstream, however, it is not required by the standard. They can be transmitted in an additional, more reliable channel, for example. The only requirement is that the parameter sets must be known to the decoder to allow the bitstream processing.
The scalable and multiview extensions of the H.264/AVC standard generally conform to the above concept. The SVC and MVC bitstreams are sequences of the NAL units similarly as the H.264/AVC bitstream. Special NAL unit types (illegal in the AVC bitstream) have been defined to carry additional data (video layers in SVC, views in MVC) introduced by these extensions.

1.2. Error Resilience Tools in H.264/AVC

Error resilience tools are available in many video coding standards. However they are limited to the frame segmentation into slices or group of blocks (GOB) in most cases. The H.264/AVC standard introduces new error resilience tools [3], [6]:

- redundant slices – additional (redundant) data are added to the normal (non-redundant) data representing the entire frame or a part of the frame;
- arbitrary slice order – the frame is divided into slices which are transmitted in non-raster (arbitrary) order;
- slice groups – macroblocks in a frame are allocated to a slice group. Six predefined allocation maps (Fig. 2) can be used, additionally explicit macroblock allocation mode is also available. This technique is also known as flexible macroblock ordering (FMO);
- bit stream partitioning – encoded slice is divided into three partitions containing respectively: slice and macroblock headers, residual data for intra coded macroblocks, and residual data for inter coded macroblocks.

These tools are available only in the Baseline or Extended profile of H.264/AVC standard. It limits their applications since most available codecs conform to the Main or High profile.
The implementations of the H.264/AVC error resilience tools are widely reported in the literature. Dynamic slice group mapping based on a macroblock classification algorithm for prioritized video transmission is presented in [7]. Combined flexible macroblock ordering (FMO) and redundant slices algorithm is presented in [8]. Multiple description coding based on redundant slices is discussed in [9]. The redundant picture coding combined with reference picture selection and reference picture list reordering method is presented in [10]. An interesting approach based on an optimal slicing and unequal error protection is proposed in [11]. Technique based on redundant pictures inserted periodically into encoded sequence is presented in [12].

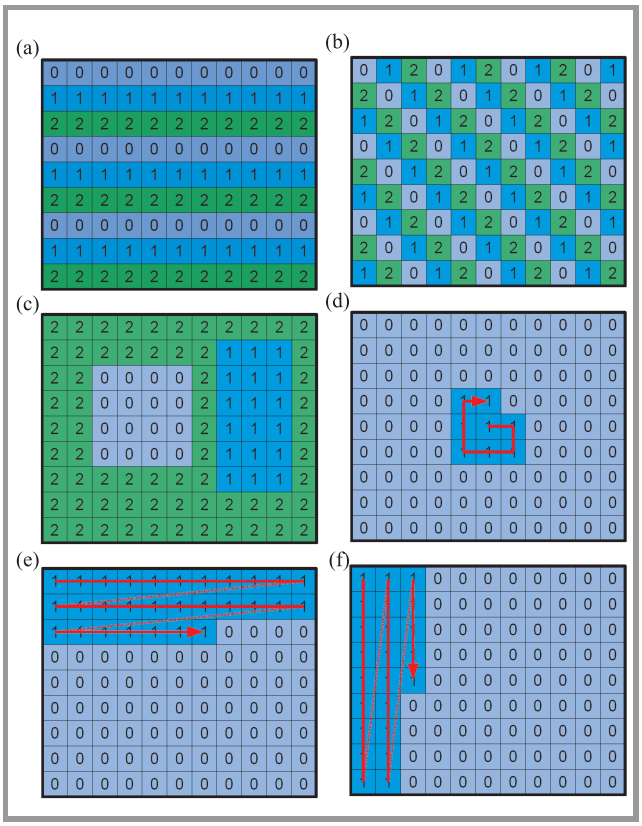

Fig. 2. Standard slice group allocation maps: (a) interleaved, (b) dispersed, (c) foreground and background, (d) box-out, e) raster scan, (f) wipe.

Error resilience tools are usually used jointly with the error concealment techniques which try to reconstruct the parts of the bitstream lost due to transmission errors. Usually perfect reconstruction is not possible. However, even if only approximation of the lost fragments can be found, the overall quality of the reconstructed sequence is improved. Two error concealment algorithms are implemented in the H.264/AVC reference software [13]: frame copy and motion vector copy [14], [15]. The first algorithm simply copies the pixel in the concealed frame from the previous decoded reference frame. The motion compensation with the motion vectors copied from the previous reference frame is used in the second algorithm. The algorithm recovering lost slices in video encoded with the FMO tool, based on the edge-directed error concealment, is presented in [16]. The FMO tool is also used in the algorithm presented in [17] to recover missing motion vectors. Many error concealment techniques utilize spatial and temporal correlation in the video sequences [18]–[20].

1.3. Video Streaming

There are generally two approaches to the media delivery over the IP networks. The first one is based on the transmission protocols utilizing TCP as a transport protocol. The file download with the use of HTTP protocol is the most obvious example. The other option is so called

HTTP progressive download. The file transmitted with the use of HTTP is split into many small fragments in this case. Each fragment is transmitted in a separate HTTP request, allowing media playback after receiving only the small part of the entire file. The most sophisticated HTTP-based solution is the adaptive progressive download [21]–[24]. The several variants of each fragment of the media file are used, each is encoded with different parameter sets, e.g., bit rate. All fragments are transmitted sequentially as in classical progressive download, however, it is possible to switch between variants at fragment boundaries. The variants are selected depending on the actual network throughput. This adaptation scheme provides uninterrupted media delivery with varying quality following the change of the network conditions. The advantage of the HTTP-based solutions is an ability to traverse firewalls so it is widely used in the Internet (e.g., YouTube). However application in real-time systems is limited due to delays introduced by the TCP transmission.

The other approach to the media delivery is based on the user datagram protocol (UDP) as a transport protocol. It is preferred in real-time applications, e.g., videoconferencing or video surveillance. There are usually very strict requirements on the transmission delay in such applications. These requirements can be fulfilled only if the UDP is used. However, since the UDP is an unreliable protocol, some datagrams may be lost, duplicated or may arrive to the destination in the wrong order. The real time protocol (RTP), accompanied by the RTP control protocol (RTCP) [25], [26], were developed to eliminate these drawbacks. The RTP provides data transport mechanism, while the RTCP is a tool for data transmission monitoring. Both protocols are most often used on the top of the UDP, however, it is possible to use them with other transport protocols too. It is worthwhile to mention that the secure enhancement of the RTP has been developed [27]. It defines the media encryption algorithm as well as media integrity and authenticity verification method.

The RTP is very universal and can be used for delivering media of different types. The RTP payload format for the delivery of H.264/AVC bitstream is presented in [6], [28]. It is based on the NAL units concept presented in the Subsection 1.1. Three encapsulation modes are specified:

– single NAL unit in the RTP packet,

– multiple NAL units in the RTP packet (aggregation mode),

– NAL unit split into multiple RTP packets (fragmentation mode).

The first mode is very simple: an entire NAL unit is inserted into the RTP packet as its payload (Fig. 3). The one-byte NAL unit header serves as the RTP payload header. The `NRI` and `TYPE` fields can be used to classify how important the payload is for the sequence reconstruction. The RTP header contains additional data describing the payload:

– `PT` – payload type identifier; certain media types have been assigned fixed identifiers [29]. The identifiers for other media types, including H.264/AVC, must be assigned dynamically, e.g., within the SDP [30] messages;

– `M` – marker bit set to 1 if the payload contains the last NAL unit in the current frame;

– `TS` – timestamp of the NAL unit carried as a payload; the clock frequency for the H.264/AVC video is equal to 90 kHz;

– `SN` – sequence number of the RTP packet; allows detection of the packet loss, duplication or incorrect order;

– `SSRC` – synchronization source identifier; each participant of the RTP session is identified by its unique identifier;

– `CSRC` – contributing source identifier; used only if mixers or translators [25] are used in the RTP session;

– `CC` – CSRC count; number of the CSRC fields in the RTP header; set to zero in most cases;

– `P` – padding flag; if set the last byte of the payload contains number of the padding bytes following the packet payload; used to increase the length of the RTP packet to the fixed value required, e.g., by the encryption algorithm.

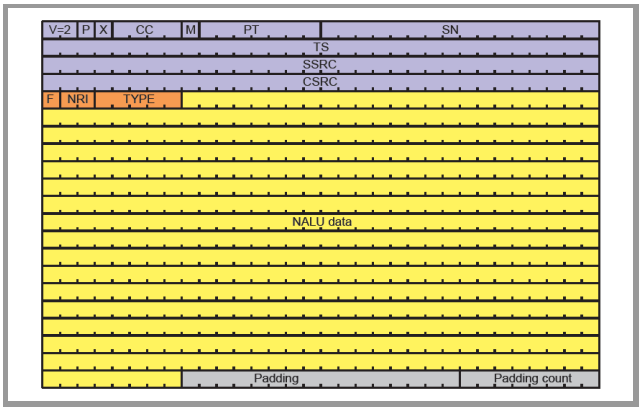

Fig. 3. Single NAL unit in the RTP packet.

Single NAL unit mode is effective if the NAL unit length fits to the network characteristic. The length of the RTP packet must not exceed the maximum length of the UDP datagram equal to 64 kB. It should also not exceed the length of the maximum transfer unit (MTU) for the given network (e.g., 1500 B for Ethernet). If the RTP packet is longer than MTU it will be fragmented by the lower layers in the IP stack. The packet fragmentation increases the probability of the packet loss.

The length of the encoded slice depends on many factors. It may easily exceed the limit of 64 kB, e.g., if the high resolution sequence is encoded with good quality (high bitrate) and no frame segmentation is used (i.e., the entire frame is encoded in one slice). In many cases it exceeds the MTU value too. The fragmentation mode provides the way to handle NAL units containing such long slices. The NAL unit is split into fragments transmitted in consecutive RTP packets. There is also an option to change the order of the NAL unit fragments. Each NAL unit fragment is appended by an additional field containing its order in this option.
The NAL units containing, e.g., parameter sets, SEI messages or encoded slices of fine fragmented frame can be very short. Their transmission in single RTP packet is ineffective due to the header overhead. The aggregation mode allows to join such short NAL units in one longer RTP packet. The aggregated RTP packet can contain either NAL units with identical timestamps or with different timestamps. Similarly, as in the fragmentation mode, NAL units do not have to be inserted into aggregated packet in its decoding order.
The payload format for the scalable extension (SVC) of the H.264/AVC is proposed in the draft specification [31]. Two modes of the SVC bitstream transmission are defined:

– single-session: all layers of the SVC stream are transmitted in a one RTP session. All packetization modes available for the H.264/AVC bitstream may be used in this mode;

– multi-session: layers of the SVC stream are transmitted in different RTP sessions. All sessions are synchronized to the same system clock. Four special packetization modes are defined for this transmission mode.

Multi-session mode is especially suitable for the multicast transmission. Separation of the SVC bitstream layers simplifies the stream adaptation to the network conditions. Specialized network devices, so called media aware network elements (MANE), can simply discard the layers which require higher throughput than is currently available.
The proposed payload format [32] for multiview extension (MVC) of the H.264/AVC is very similar to the specification for the SVC. The views contained in the MVC bitstream can be transmitted in either one RTP session (single-session mode) or in multiple synchronized RTP sessions (multi-session mode).

2. Video Streaming Framework Overview

The following requirements for the framework were specified:

– streaming of the H.264/AVC encoded video with the use of RTP/RTCP;

– monitoring and visualisation of the network parameters during transmission;

– acquisition and real-time encoding of the analogue video signal;

– decoding of the H.264/AVC stream and displaying of the reconstructed video in real time.

The open source software has been extensively used in the framework development. The framework is running under the Linux operating system. It has been written mostly in the C++ programming language, some code fragments directly interfacing with underlying libraries have been written in C. Video4Linux2 (V4L2) [33] application programming interface has been used for video capture. The graphical user interface has been created with the use of Qt library [34]. The classes from the Qwt library [35] have been used to create diagrams for network parameters visualisation. The open source library JRtpLib [36] has been used to sent and receive RTP packets and handling of the RTCP messages. The FFmpeg library [37] has been used for the H.264/AVC stream decoding. The H.264/AVC parsers have been based on the H.264 reference software [13]. The x264 [38] library has been used to encode video.
The most important classes of the framework are:

– `NalUnit` – represents the NAL unit and its timing information;

– `AnnexBReader` – parser for the H.264/AVC bitstream stored in the Annex B [1] format;

– `JmRtpReader` – parser for the H.264/AVC bitstream stored in the RTP format defined in the H.264 reference software;

– `AnnexBWriter` – stores H.264/AVC bitstream in the Annex B format;

– `JmRtpWriter` – stores H.264/AVC bitstream in the JM/RTP format;

– `Rfc3984Packetizer` – encapsulates NAL units in the RTP packets in compliance with the RFC 3984 [28];

– `Rfc3984Depacketizer` – restores NAL units encapsulated in the RTP packets;

– `RtpStreamer` – creates RTP/RTCP session for sending NAL units;

– `RtpReceiver` – creates RTP/RTCP session for receiving NAL units;

– `V4LConfigWidget` – configures the V4L2 video capturing device;

– `V4LStreamerThread` – streams video from the capturing device to the memory buffers;

– `X264ConfigWidget` – configures the x264 encoder;

– `X264EncoderThread` – encodes video stored in the memory buffers;

– `Decoder` – decodes H.264/AVC bitstream and converts decoded YUV frames into RGB images.

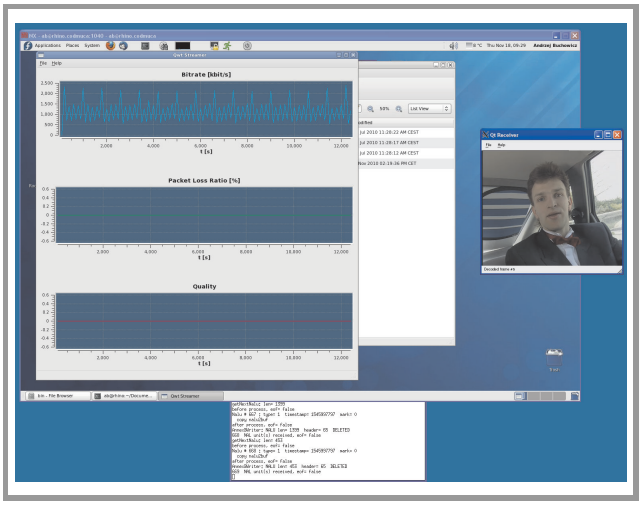

Fig. 4. Screenshot showing two framework applications. The H.264/AVC streamer is shown on the left, diagrams displays bit rate, packet loss ratio and a quality measure for the reconstructed video. The decoded H.264/AVC stream is displayed on the right.

The framework (Fig. 4) has been compiled and tested on the Fedora 10/12/14 and Ubuntu 10.10 distributions. The framework has been developed with the use of standard libraries and development tools so it should be possible to use it on other Linux distributions too. Adaptation for the other operating systems will require the complete rewrite of the classes responsible for video capture.

3. Experimental Results

The framework presented in the previous section has been used for analysis, development and testing of video streaming techniques. The comparison of the H.264/AVC error resilience techniques will be presented as an example of the experimental results obtained with the framework.
The CIF resolution *Carphone* test sequence has been encoded by the H.264/AVC reference software encoder [13] configured for the Baseline profile [1]. Group of pictures composed of 12 I/P frames and a constant value of the quantization parameter QP have been used. Three frame segmentation modes have been used: an entire frame in one slice (denoted as frame), slices containing one row of macroblocks (row) and slices with the length not exceeding the 1400 B which is less then the MTU value. Additionally two slice group modes have been used: interleaved (inter) and dispersed (disp). The rate-distortion (R-D) curves for the selected coding parameters are presented in Fig. 5. The error resilience tools reduce the coding efficiency, especially if the row slices segmentation or the dispersed slice group is used.

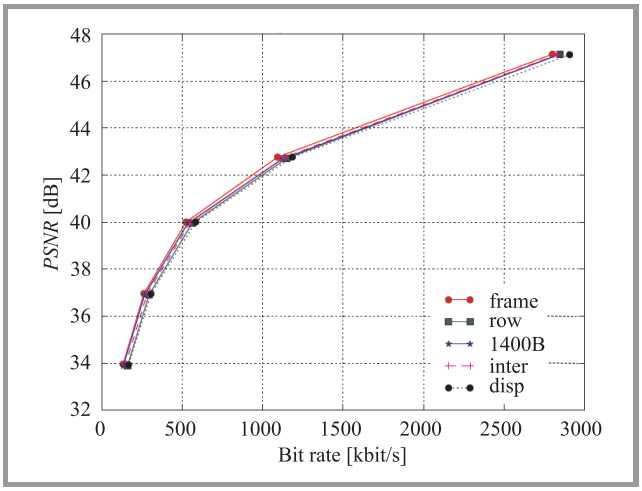

Fig. 5. The R-D curves for coding parameters.

The test sequences encoded with the $QP = 30$ (bit rate $250 \div 300$ kbit/s depending on coding parameters) have been streamed over IP network with controlled throughput. Single NAL unit in the RTP packet has been used in all experiments. Each sequence have been transmitted 5 times for selected network throughputs. The averaged packet loss ratio (PLR) is shown in Fig. 6. The PLR is higher for the row slices segmentation mode than for any other mode.

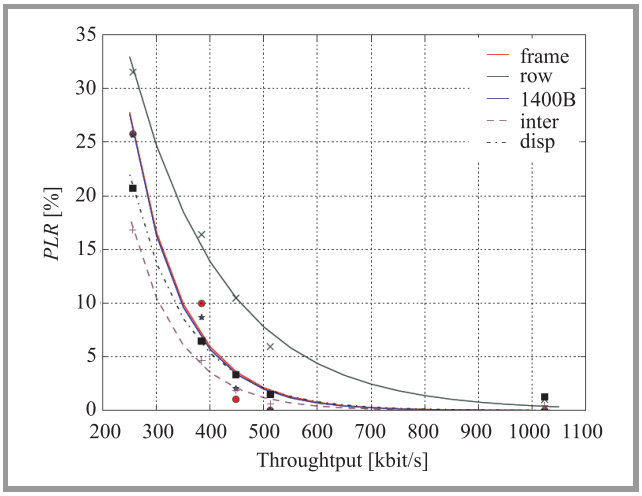

Fig. 6. The PLR for coding parameters.

The received bitstreams have been decoded by the H.264/AVC reference software decoder [13]. The peak signal-to-noise ratio (PSNR) values for each decoded bitstream have been calculated. If the bitstream has not been decoded due to transmission errors it has been assumed that $PSNR = 0$ dB. The averaged PSNR values without any error concealment in the decoder are shown in Fig. 7. The most effective is frame segmentation into slices shorter than MTU, slice group modes are slightly better than coding the entire frame in one slice.
The effectiveness of the slice group increases if the error concealment techniques are used in the decoder.

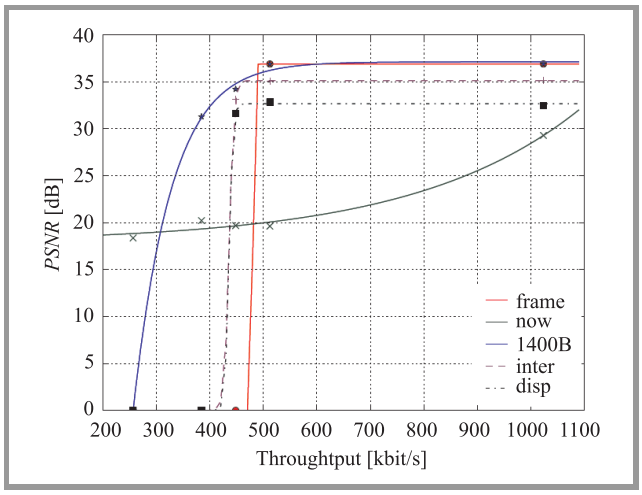

Fig. 7. Averaged PSNR with no error concealment in the decoder.

Figures 8 and 9 present the averaged PSNR for the frame copy and the motion copy error concealment mode respectively.

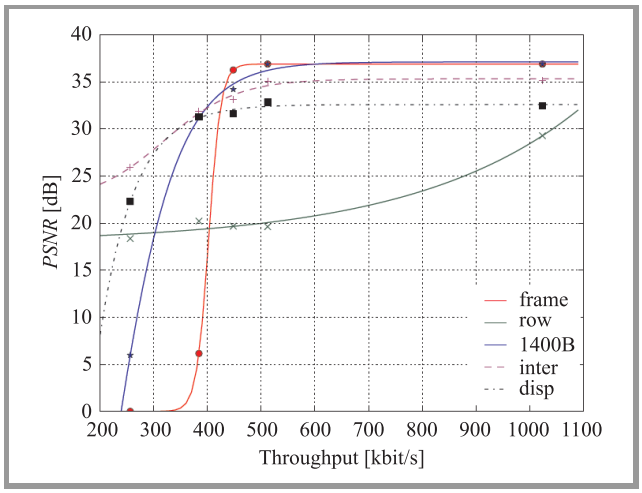

Fig. 8. Averaged PSNR for the frame copy error concealment mode.

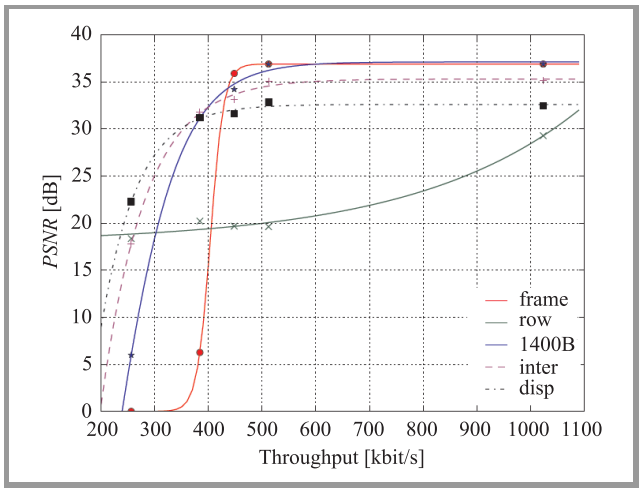

Fig. 9. Averaged PSNR for the motion copy error concealment mode.

The experimental results show that slice groups – new error resilience tool available in the H.264/AVC standard can improve the effectiveness of the transmission if the error concealment techniques are used in the decoder. However, the proper NAL unit encapuluation mode must also be selected. The lengths of NAL units in the test sequences selected for the experiment do not exceed the MTU value in most cases. Therefore, the single NAL unit in the RTP packet has been used. The results for other test sequences, with longer NAL units, would be different. The fragmentation mode would have to be used to achieve comparable effectiveness. It is worthwhile to mention, that frame segmentation into slices of length not exceeding the MTU value provides high effectiveness even if no error concealment algorithms are used in the decoder. This frame segmentation mode is available in all profiles of the H.264/AVC standard and it can always be used with the single NAL unit packetization mode.

4. Conclusions

The framework presented in this paper is a tool for testing video streaming techniques. It is based on the open source software, the H.264 reference software is also used. The framework allows streaming of the H.264/AVC video with the use of RTP/RTCP. The preencoded video stored in the file or real-time encoded video from capturing device can be transmitted. The received video can be decoded and displayed in real-time or stored in the file for further processing. The transmission parameters: bit rate, packet loss ration can be continuously displayed. The framework can be easily extended for other codecs and transmission protocols. It has been developed for the Linux operating system, but most of the libraries are portable, so the adaptation for other operating systems is possible.

References

[1] *Information technology – Coding of Audio-Visual Objects – Part 10: Advanced Video Coding*, ISO/IEC 14496-10, 2008

[2] Y. Q. Shi, H. Sun, *Image and Video Compression for Multimedia Engineering. Fundamentals, Algorithms, and Standards.n*, CRC Press, 2008.

[3] I. E. Richardson, *The H.264 Advanced Video Compression Standard*, Wiley, 2010.

[4] A. Kondoz, *Visual Media Coding and Transmission*, Wiley, 2009.

[5] *Information technology – Generic Coding of Moving Pictures and Associated Audio Information: Video*, ISO/IEC 13818-2, 2000.

[6] S. Wenger, "H.264/AVC over IP", *IEEE Trans. Circ. Sys. Video Technol.*, vol. 13, no. 7, pp. 645–656, 2003.

[7] S. K. Im and A. J. Pearmain, "Error resilient video coding with priority data classication using H.264 exible macroblock ordering", *IET Image Proces.*, no. 1, vol. 2, pp. 197–204, 2007.

[8] B. Katz, S. Greenberg, N. Yarkoni, N. Blaunstien, and R. Giladi, "New error-resilient scheme based on FMO and dynamic redundant slices allocation for wireless video transmission", *IEEE Trans. Broadcast.*, vol. 53, no. 1, pp. 308–319, 2007.

[9] T. Tillo, M. Grangetto, and G. Olmo, "Redundant slice optimal allocation for H.264 multiple description coding", *IEEE Trans. Circ. Sys. Video Technol.*, vol. 18, no. 1, pp. 58–70, 2008.

[10] C. Zhu, Y.-K. Wang, M. M. Hannuksela, and H. Li, "Error resilient video coding using redundant pictures", *IEEE Trans. Circ. Sys. Video Technol.*, vol. 19, no. 1, pp. 9–70, 2009.

[11] O. Harmanci and A. M. Tekalp, "Rate-distortion optimal video transport over IP allowing packets with bit errors", *IEEE Trans. Image Proces.*, vol. 16, no. 5, pp. 1315–1326, 2007.

[12] I. Radulovic, P. Frossard, Y-K. Wang, M. M. Hannuksela, and A. Hallapuro, "Multiple description video coding with H.264/AVC redundant pictures", *IEEE Trans. Circ. Sys. Video Technol.*, vol. 20, no. 1, pp. 144–148, 2010.

[13] K. Suhring, *H.264/AVC JM Reference Software* [Online]. Available: http://iphome.hhi.de/suehring/tml

[14] A. M. Tourapis, A. Leontaris, K. Shring, and G. Sullivan, *H.264/14496-10 AVC Reference Software Manual*, JVT-AE010, 2009.

[15] K.-P. Lim, G. Sullivan, and T. Wiegand, *Text Description of Joint Model Reference Encoding Methods and Decoding Concealment Methods*, JVT-X101, 2007.

[16] M. Ma, O. C. Au, S.-H. G. Chan, and M.-T. Sun, "Edge-directed error concealment", *IEEE Trans. Circ. Sys. Video Technol.*, vol. 20, no. 3, pp. 382–395, 2010.

[17] J. Wu, X. Liu, and K.-Y.Yoo, "A temporal error concealment method for H.264/AVC using motion vector recovery", *IEEE Trans. Consumer Electron.*, vol. 54, no. 4, pp. 1880–1885, 2008.

[18] G.-L. Wu, C.-Y. Chen, T.-H. Wu, and S.-Y. Chien, "Efficient spatial-temporal error concealment algorithm and hardware architecture design for H.264/AVC", *IEEE Trans. Circ. Sys. Video Technol.*, vol. 20, no. 11, pp. 1409–1422, 2010.

[19] K. Seth, V. Kamakoti, and S. Srinivasan, "Efficient motion vector recovery algorithm for H.264 using B-spline approximation", *IEEE Trans. Broadcast.*, vol. 56, no. 4, pp. 467–480, 2010.

[20] X. Chen, Y. Y. Chung, C. Bae, X. He, and W.-C. Yeh, "An efficient error concealment algorithm for H.264/AVC using regression modeling-based prediction", *IEEE Trans. Consumer Electron.*, vol. 56, no. 4, pp. 2694–2701, 2010.

[21] R. Pantos and W. May, *HTTP Live Streaming*, 2010 [Online]. Available: http://tools.ietf.org/html/ draft-pantos-http-live-streaming-04

[22] A. Zambelli, *IIS Smooth Streaming Technical Overview*, Microsoft Corporation, 2009.

[23] *Adobe – HTTP Dynamic Streaming* [Online]. Available: http://www.adobe.com/products/httpdynamicstreaming/

[24] *Information Technology – MPEG Systems Technologies – Part 6: Dynamic Adaptive Streaming over HTTP (DASH), ISO/IEC FCD 23001-6*, ISO/IEC JTC 1/SC 29/WG 11, w11749, 2011.

[25] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Application*, RFC 3550, 2003.

[26] C. Perkins, *RTP Audio and Video for the Internet*. Boston: Addison-Wesley, 2006.

[27] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman, *The Secure Real-time Transport Protocol (SRTP)*, RFC 3711, 2004.

[28] S. Wenger, M. M. Hannuksela, T. Stockhammer, M. Westerlund, D. Singer, *RTP Payload Format for H.264 Video*, RFC 3984, 2005.

[29] H. Schulzrinne and S. Casner, *RTP Proile for Audio and Video Conferences with Minimal Control*, RFC 3551, 2003.

[30] M. Handley, V. Jacobson, and C. Perkins, *SDP: Session Description Protocol*, RFC 4566, 2006.

[31] S. Wenger, Y.-K. Wang, T. Schierl, A. Eleftheriadis, *RTP Payload Format for SVC Video* [Online]. Available: http://tools.ietf.org/ html/draft-ietf-avt-rtp-svc-21

[32] Y.-K. Wang, T. Schierl, *RTP Payload Format for MVC Video* [Online]. Available: http://tools.ietf.org/html/draft-wang-avt-rtp-mvc-05

[33] *Linux Media Infrastructure API* [Online]. Available: http://www.linuxtv.org/ downloads/v4l-dvb-apis

[34] *Qt – A cross-platform application and UI framework* [Online]. Available: http://qt.nokia.com/products

[35] *Qwt – Qt Widgets for Technical Applications* [Online]. Available: http://qwt.sourceforge.net

[36] *Jori's page – CS/Jrtplib* [Online]. Available: http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jrtplib

[37] *FFmpeg* [Online]. Available: http://www.ffmpeg.org

[38] *VideoLAN – x264, the best H.264/AVC Encoder* [Online]. Available: http://www.videolan.org/developers/x264.html

Andrzej Buchowicz received the M.Sc. degree in Electronics and Ph.D. degree in Telecommunication from the Faculty of Electronics, Warsaw University of Technology in 1988 and 1997, respectively. He is currently an Associate Professor in the Institute of Radioelectronics, Warsaw University of Technology. His current research interest include video coding, streaming and adaptation.
E-mail: A.Buchowicz@ire.pw.edu.pl
Institute of Radioelectronics
Warsaw University of Technology
ul. Nowowiejska st 15/19
00-665 Warsaw, Poland

Grzegorz Galiński received his M.Sc. in Electronics in 1997 and Ph.D. in 2003 from Warsaw University of Technology, Poland. Since 2002 he is with Institute of Radioelectronics at Warsaw University of Technology. His main interests include: image and video compression, multimedia indexing and multimedia systems.
E-mail: G.Galinski@ire.pw.edu.pl
Institute of Radioelectronics
Warsaw University of Technology
Nowowiejska st 15/19
00-665 Warsaw, Poland