Proc. of Int. Conf. on Advances in Mobile Computing and Multimedia (MoMM), Yogyakarta, Indonesia, Dec. 2006.

Performance of a H.264/AVC Error Detection Algorithm Based on Syntax Analysis

Luca Superiori*, Olivia Nemethova* and Markus Rupp*

Abstract. *In this work we investigate the possibility of detecting errors in H.264/AVC encoded video streams. We propose a method for the detection of errors exploiting the variable length code codewords, range and significance of the H.264/AVC information elements. We evaluate the performance of such syntax analysis based detection technique for different bit error probabilities and compare it to the typical packet discard approach. Focus is given on low rate video sequences.*

1 Introduction

H.264/AVC (Advanced Video Coding) [1] is the newest video coding standard, written by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG) as the product of a collective partnership effort known as the Joint Video Team (JVT). This standard is especially suitable for low data rate applications as it provides substantially better video quality at the same data rates compared to previous standards (MPEG-2, MPEG-4, H.263), with only a moderate increase of the complexity. Moreover, H.264/AVC was designed to support a wide variety of applications and to operate over several types of networks and systems.

*V*ideo telephony and video streaming over IP packet networks are quite challenging application due to their requirement on delay and data rates. A video stream is encoded and packetized in Real Time Protocol (RTP) packets. These packets are then transported end-to-end within User Datagram Protocol (UDP). Unlike Transmission Control Protocol (TCP), UDP does not provide any retransmissions control mechanism. Nevertheless, it has been widely adopted for video streaming and video telephony, since the end-to-end retransmissions would cause unacceptable delays. Thus, in such real-time applications, transmission errors cannot be completely avoided.

To allow for applications even in error-prone environments like mobile networks, apart from the improved compression performance, H.264/AVC provides several error resilience features. Therefore, the 3rd Generation Partnership Project (3GPP), standardizing the Universal Mobile Telecommunications Network (UMTS), has approved the inclusion of H.264/AVC as an optional feature in release 6 of its mobile multimedia telephony and streaming services specifications ([2], [3]).

To facilitate error detection at the receiving entity, each UDP datagram is provided with a simple 16 bit long checksum. The packets with detected errors are typically discarded and missing parts of video are subsequently concealed. The reason for this handling is the variable

*Institute of Communications and Radio-Frequency Engineering, Vienna University of Technology, Austria, Gusshausstrasse 25/389, A-1040 Vienna, Austria, {lsuper,onemeth,mrupp}@nt.tuwien.ac.at

length coding (VLC). H264/AVC implements context adaptive VLC (CAVLC) in its baseline profile. After a bit error, (CA)VLC may easily desynchronize, making the correct distinguishing between the following codewords impossible. Therefore, without any resynchronization mechanism and/or additional detection/decoding mechanism (e.g. [4], [5], [6]), the decoding of such stream may result in considerable visual impairments, or may become even impossible (due to the non-existing code-words, too many or too few bits left for decoding). The detection of errors allows for utilization of correctly received parts of the packet for the decoding. Since a packet usually contains rather large picture area, it may considerably improve the quality of reconstruction at the receiver. The structure of the bit stream — the syntax of its information elements — may also provide some means to detect errors. For H.263 codec, the performance of a simple syntax check method was evaluated in [7]. However, the structure of the H.264/AVC bitstream and the CAVLC differs considerably from the structure and VLC of the H.263 bitstream.

In this work we investigate the possibility of detecting errors in H.264/AVC encoded video stream. We propose a method for detection of errors exploiting the codewords, range and significance of the H.264/AVC information elements. We evaluate its performance and compare it to the typical packet discard approach. The focus is given on the baseline profile (targeting the video conferencing, streaming and especially mobile applications) and thus, we work with CAVLC rather than with context adaptive binary arithmetic coding (CABAC) designed mainly for the storage applications. We do not take into account detection of errors within the RTP/UDP/IP header. Errors within the header could also be detected by other means, e.g. UDP-lite [8], or using the information from lower layers depending on the underlying system. This paper is organized as follows. Section 2 introduces briefly the architecture of H.264/AVC codec. The structure of the H.264/AVC RTP bitstream is described in Section 3. Section 4 analyzes individual information elements and presents the way in which their syntax may be used to detect errors. Results of the performance evaluation and comparison with alternative methods are provided in Section 5. Section 6 contains conclusions and some final remarks.

2 H.264/AVC Design Characteristics

This section describes the fundamentals of the H.264/AVC architecture. Similarly to the other hybrid block based coders, the H.264/AVC encoding process follows a hierarchical structure. The whole video is segmented into Groups of Pictures (GOP), each of them contains a certain number of still pictures called frames. H.264/AVC allows for three frame types: Intra (I) frame, encoded exploiting spatial correlation, predicted (P) frames, encoded using reference to previous frames, and bi-directionally predicted (B) frames, exploiting temporal correlation between the current frame and the future or previous ones. In transmissions over error prone channels, I frames are used to refresh the sequence and to reduce error propagation in time. The length of a GOP is defined as the distance between two consecutive I frames. Each frame is subdivided into slices and each slice in macroblocks (MB). The dimension of a slice depends on the encoding properties, H.264/AVC allows slice dimension expressed in number of macroblocks or in bytes. In the latter case, the number of slices per frame depends on the frame type and on the frame content.

H.264/AVC is conceptually separated into a VCL (Video Coding Layer) and a NAL (Network Abstraction Layer). VCL is responsible for the core block-based hybrid coding functions, NAL provides network friendliness by allowing the output bit-stream for being transmitted over different transport layer protocols. The video data is transformed in a stream of information units called NALU (Network Abstraction Layer Unit) and further encapsulated into different protocols. Fig. 1 shows an IP packet for a common packet oriented transmission environment.

NALUs are subdivided into non-VCL and VCL NAL units. Non-VCL NALUs contain set of video parameters. To this category belong Sequence Parameter Set (SPS), defining profile, resolution and other properties of the sequence, and Picture Parameter Set (PPS), containing type of entropy coding, slice group and quantization properties. VCL NALUs contain the data associated to the video slice, each VLC NALU refers to a non-VCL NALU as shown in Fig. 2. In this work, we will docus on syntax check performed on VLC NALUs, since non-VCL NALUs would not be transmitted within the RTP payload, but provided in the SDP (Session Description Protocol) ([3], [2]).

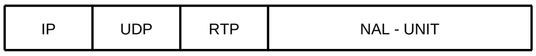


Figure 1. NAL unit encapsulation

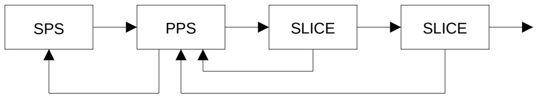


Figure 2. NAL unit sequence

3 H.264/AVC Bitstream Structure

This section offers a brief overview over the H.264/AVC syntax, as produced by the JM (Joint Model) reference software [9] and described in [1]. The encoded video, outputted as raw bit-stream, is segmented into basic elements stored in the NAL payload. By attaching a one byte header, a *NAL unit* is obtained. The NAL header consists of a *forbidden bit* F, a NRI (NAL Reference Identification) field, signalizing the importance of the NAL unit, and a TYPE parameter specifying one of the currently defined NALU types.

The standard [1] defines different coding strategies for each of the syntax fields. Elements above the slice layer are encoded by means of fixed or variable length coding. In baseline profile the CAVLC is used at the slice layer and below. CAVLC maps each codeword to a different table depending on the data statistics. A variable length coding, by assigning shorter codewords to the most frequent symbols, offers advantages in term of bit-rate saving but, on the other hand, an erroneous bit can cause desynchronization.

4 Proposed Mechanism

Syntax check limits in H.263, discussed in [7], are in H.264/AVC even more evident. Entropy coding strategies and lack of synchronization words between macroblocks make the error usually not detectable immediately. Therefore, the error propagates until the end of the slice. Since most of the codewords are entropy encoded and self-decodable (without necessity of a look-up table), it is quite unusual to spot illegal codewords by reading the bitstream. Thus, a contextual analysis of each element is needed.

4.1 VCL NALU structure

The examined bitflows refer to different files obtained by encoding a QCIF (176 × 144 pixels) video sequence using JM codec in baseline profile. Since we want to simulate the effects of errors appearing in packet network applications, the chosen file mode is RTP. The Fig. 3 shows the structure of the considered VCL NALU payload, composed by a Slice Header (SH) and several macroblocks. The slice header contains the basic characteristic of the slice, error affecting one of its parameter could make the whole slice undecodable.
The H.264/AVC decoder consists basically of two blocks shown in Fig. 4. The *READ* phase deals with the reading of the raw bitstream and the partitioning in different codewords. During the *DECODE* those codewords are interpreted as information elements (IE) and used to reconstruct the encoded slice.

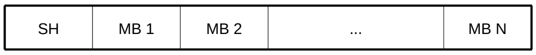


Figure 3. Structure of a VCL NAL unit

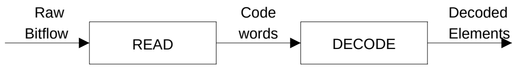


Figure 4. Conceptual parameter decoding

the standard defines a type of encoding for each parameter. Without loss of information we decided to divide them in four groups.

- Fixed Length codewords — *FL*: Words composed by a number of bits known a priori.
- Exp-Golomb coded codewords — *EG*: Exponential Golomb Codes, adopted by H.264/AVC, are characterized by a regular logical structure consisting of a predetermined code pattern and no requirement of decoding tables. Each exp-Golomb codeword embodies the following elements:

$$\underbrace{0_1 \ldots 0_M}_{M} 1 \underbrace{b_1 \ldots b_M}_{M}.$$

  The first M zeros and the middle one are regarded as *prefix* while the following M bits represent the *info* field. The exp-Golomb field *codeNum* is obtained as

$$\text{codeNum} = 2^M + \text{info} - 1.$$

  This information is then used to obtain the encoded value depending on the chosen exp-Golomb coding style (unsigned, signed and truncated). An error affecting the leading zeros or the middle 'one' affects the decoding by modifying the value of M, therefore causing the desynchronization of the decoding. An error in the *info* field causes deviation of the decoded parameter values and, possibly, affects the following elements, but without direct desynchronization.
- Tabled codewords — *TE*: This category incluedes the VLC words to be found in a look-up table. H.264/AVC defines several VLC tables for different syntax elements and contexts.
- VLC level codewords — *VL*: This is the context adaptive coding style, characteristic for the residual levels encoding. Each residual is encoded using a VLC-N procedure, where N is a parameter depending on the value of the previous decoded levels. The standard defines integer values of N in the range [0,6]. For VLC-0 the codeword has the following structure:

$$\underbrace{0_1 \ldots 0_M}_{M} 1,$$

containing both absolute value and sign of the level. For higher N the VLC-N has the form:

$$\underbrace{0_1 \ldots 0_M}_{M} 1 \underbrace{i_1 \ldots i_{N-1}}_{N-1} s.$$

Similarly to exp-Golomb codes, the codeword starts with a sequence of M leading zeros followed by a one, an *info* field consisting of *N-1* bits and sign bit s. The encoded value is then obtained as

$$(-1)^s \cdot ((M+1) << (N-1) + 1 + \text{info}),$$

where << represent the bitwise shift operation.
The VLC-0 codewords are highly susceptible to errors since the whole information is contained in the leading zeros. For VLC-N the first *M+1* bits are critical, errors lying in the *info* field or sign do not cause desynchronization of the but affect only the decoded level. While errors in the *info* field could cause the use of a false VLC procedure for the following decoded items, errors in the sign are not detectable at all.

4.2 Syntax Check Rules

A first error analysis consists of subdividing detectable errors in different sets depending on their characteristics.

- **Illegal Codeword — IC**: Arises when the codeword does not find correspondence in the appropriate look-up table. IC occurs during the *READ* process for tabelled, exp-Golomb coded and fixed length codewords.
- **Out of Range Codeword — OR**: Results when the decoded values lie outside the legal range. It appears during *READ* process for all types of codewords. The decoded parameter can only take values between $[0,K]$, an error is produced if the read value is greater than K.
- **Contextual Error — CE**: Occurs when the decoded word leads the decoder to illegal actions. It arises during *DECODE* phase for tabelled, exp-Golomb coded and fixed length encoded parameters.

The presented errors are not strictly related to the actual bitstream failure. They are rather referred to the detected anomalies caused by propagation of previously undetected errors.

Once the encoding techniques and the error categories have been presented, we can describe the syntax check applied on the parameter of a common H.264/AVC stream.

Using the notation of the output trace file generated by the JM 10.2 (corresponding to the one of the standard [1]), in the following subsections the parameters encoded in I slices, P slices and slice headers are presented. For each of the considered field the encoding style is outlined and, where useful, a brief investigation of the error characteristics is performed.

I Frames

Parameter Name	Enc.	Err.
`mb_type`	*EG*	***OR***
`intra4x4_pred_mode`	*FL*	***CE***

The intra 4x4 prediction mode is encoded by a codeword of fixed length of one (1) or 4 ($0b_1b_2b_3$) bits. An error is detectable only by a contextual analysis. Spatial prediction uses reference to the surrounding macroblocks, if they are not available, not already decoded or belonging to another slice, a contextual error is produced.

`intra_chroma_pred_mode` *EG* ***OR***

`coded_block_pattern` *EG* ***OR***

`mb_qp_delta` *EG* ***OR***

`Luma(Chroma) # c & tr.1s` *TE* ***IC***

The look-up table is not complete. The decoded codeword could not be associated to any legal value.

`Luma(Chroma) trailing ones sign` *EG*

The signs of the trailing ones are fixed length encoded and do not influence any of the following parameters. By means of syntax check it is not possible to detect such errors.

`Luma(Chroma) lev` *VL* ***OR/CE***

Decoded pixels can only assume values lying in the range [0,255]. During *reading*, values outside the bounds are immediately associated to errors. During *decoding* phase, the residuals are added to the predicted values and the check is repeated. An extended range $[-\lambda, 255+\lambda]$ is considered due to quantization offset.

`Luma(Chroma) totalrun` *TE* ***IC***

`Luma(Chroma) run` *TE* ***IC/OR***

Depending on the number of remaining zeros, a VLC tables is chosen. For more than six remaining zeros a single table covering the zero run range [0,14] is used. Therefore, the decoder is exposed to out of range errors.

P Frame

`mb_type` *EG* ***OR***

See I-frames

`mb_skip_run` *EG* ***OR/CE***

The number of macroblocks within a frame is known. The number of skipped macroblocks can not be greater than the total number of MBs minus the number of the already decoded MBs

`sub_mb_type` *EG* ***OR***

`ref_idx_l0` *EG* ***OR/CE***

The index of the reference frame can not be greater than the actual reference list size

`mvd_l0` *EG* ***OR***

Slice Header

`first_mb_in_slice` *EG* ***OR***

`pic_parameter_set_id` *EG* ***OR/CE***

The PPS index can not be greater than the number of defined PPS

`slice_type` *EG* ***OR***

`frame_num` *EG* ***OR***

Depending on the GOP structure an out of range error could be detected

`pic_order_cnt_lsb` *EG* ***OR***

`slice_qp_delta` *EG* ***OR***

4.3 Error Handling

In this work three error handling strategies have been tested. We focused our analysis on the detecting performance rather than on the concealment results. Detected errors are concealed by means of a *copy-and-paste* algorithm, replacing each corrupted macroblock with the spatially corresponding one in the previous frame.

4.3.1 Straight Decoding — *SD*

Using this handling strategy the bitstream is decoded without any concealment function. Each erroneous parameter value is replaced with the most similar legal one. Fig. 5(a) shows a corrupted frame decoded using the error handling strategy *SD*. The different slices are separated by means of the semitransparent dark lines. An error is inserted into macroblock 24 (red square). Since no concealment method is called, the rest of the slice is decoded using desynchronized VLC codewords.

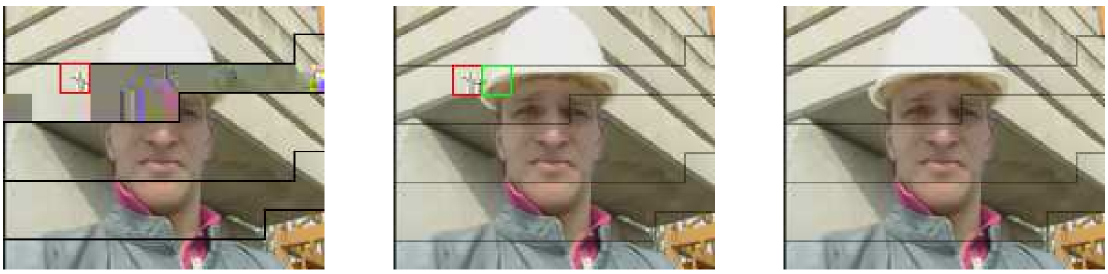
4.3.2 Macroblock level concealment — *MBLC*

In this handling approach, once an error has been detected, the current macroblock and the following one are concealed until the end of the slice. Fig. 5(b) displays the same corrupted frame as the previous example. Using *MBLC*, once in macroblock 25 (green square) the error is detected, the macroblocks from 25 up to the end of slice (38) are concealed by means of copy-and-paste algorithm.

Further analysis shows that conceptual errors arise also at slice level. Due to desynchronization, the code length could be insufficient or overmuch. If there are not enough parameters to decode the whole slice, a concealment method is called for the remaining macroblocks. On the other hand, if the *first_mb_in_slice* parameter of a slice is lower than the number of previously decoded macroblocks, an error is produced and the exceeding macroblocks are overwritten.

4.3.3 Slice level concealment — *SLC*

This strategy relies on the checksum information provided by UDP protocol. If there is an error detected within the slice, the entire slice is concealed. This is illustrated in Fig. 5(c).



(a) SD (b) MBLC (c) SLC

Figure 5. Corrupted frame decoding

5 Performance Analysis

To evaluate performance of the three presented error handling methods, we performed experiments with corrupted sequences. For all three strategies the same decoder was used.

5.1 Experimental Setup

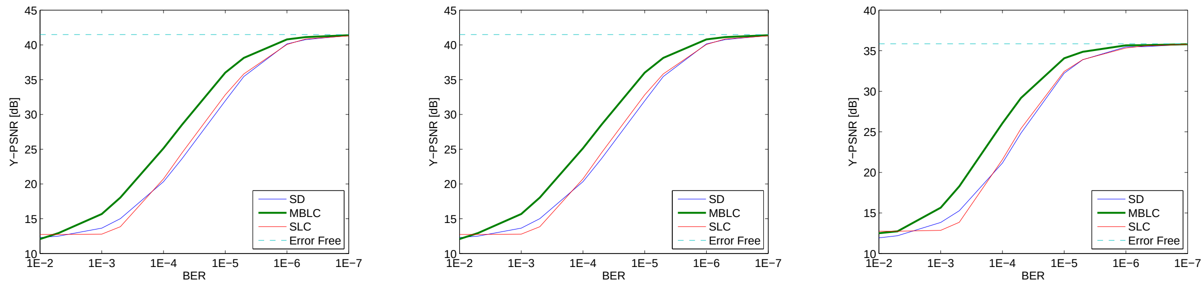
For our experiments we used the encoder and decoder of Joint Model H.264/AVC v.10.2 [9] adapted to our needs by introducing the following additions:

- The decoder is able to read extern text files containing the error pattern and to modify the bits to be corrupted at NAL level.
- When a read codeword or a decoded parameter takes an illegal value, its value is restored to the most similar legal one. Addittionally two error flags, one at macroblock level and one at slice level, are forced to one.
- All resilience methods integrated in JM, both in macroblock and slice level, are disabled.

We used the "Foreman" sequence with QCIF resolution, having total length of 400 frames and played at 30 frames per second. We encoded the sequence using baseline profile (no B frames and no CABAC are used) and GOP size was 10. We chose slicing mode with maximum number of bytes per slice limited to 700 and number of reference frames for motion estimation set to five.
The tests were performed by corrupting and decoding the sequences encoded with different quantization parameters. Errors were randomly generated with various bit error rates (BER). The simulation were performed on a set of 75 sequences per BER per QP.

5.2 Results

To evaluate the improvement of the end-to-end video quality, we use the peak signal-to-noise ratio of the luminance component (Y-PSNR) as outputted by JM. As a reference sequence we used the *non-compressed* original (non-degraded) sequence.
Each of the graphs plotted in Fig. 6 shows the trend of Y-PSNR for the three proposed error handling strategies for three different quantization parameters. The values are obtained by averaging over Y-PSNR of all frames of all sequences decoded with the same BER.



(a) QP = 20 (b) QP = 24 (c) QP = 28

Figure 6. Performance of the different error handling strategies

As expected, the *SD* error handling offers very low video quality since no error concealment routine is called and the artifacts are quite annoying, as shown in Fig. 5(a). Macroblock level concealment, based on syntax check, outperforms the slice level concealment. This is quite surprising, because the undetected errors result in artifacts, that are not concealed. These, however, remain local if the error is detected in some successive information elements, which offers a good compromise between error detection capabilities and error concealment efficiency.

The decoding of a slice with an error at the position b by using method *MBCL* can be described by three intervals as shown in Fig. 7:

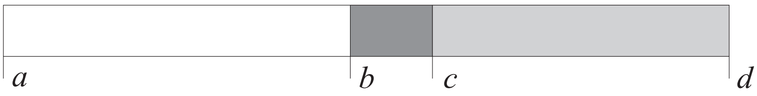
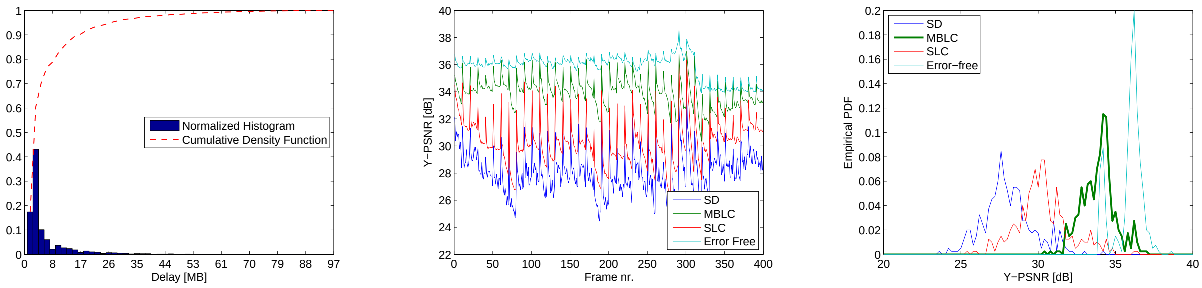


Figure 7. Origination of macroblock level concealment delay

- ***Interval [a,b):*** The slice is correctly decoded from its begin a up to the error at the position b.
- ***Interval [b,c):*** The error is undetected until the position $c \geq b$. This part is decoded incorrectly.
- ***Interval [c,d]:*** Starting from the position c until the end of the slice d concealment is used.

The efficiency of the proposed approach lies in the interval *[a,b)* where the decoding is performed properly. The improvement is even higher in the sequences with faster motion, resp. in the sequences with reduced frame rate, since in such cases the performance of error concealment methods decreases. The macroblocks decoded in the *[b,c)* interval cause errors with higher magnitude. However, Fig. 8(a) shows the normalized histogram of the delay, expressed in macroblocks, between the error occurrence and the error detection. Since over 65% of the detections happen within 2 macroblocks and the 85% within 10 macroblocks, the effect of the artifacts in *[b,c)* is limited. Since P frames are usually encoded using few slices, *SLC* performs poorly particularly for this frames. Thus, discarding the entire slice causes large errors, both for the frame where the error occurs and for the following ones. The detection probability at slice level was evaluated. Simulations were performed by inserting a bit in random position in each slice. Over the 57% of the errors were detected.

In Fig. 8(b) the average Y-PSNR over time (frame number) is shown for the three investigated methods. Averaging was performed over all sequences decoded with a BER of 10^{-5}. Another visualization of the reconstruction quality for the same BER can be seen in Fig. 8(c).



(a) Detection delay (b) Y-PSNR over time (c) Empirical PDF

Figure 8. Detection delay and quality comparison

The quality improvement achieved by syntax analysis compared to the packet discard strategy is approximately 4 dB which is quite significant from the user experienced quality point of view.

6 Conclusions

Three possible error handling strategies are compared in this paper: common packet discard approach, strategy based on straight decoding of H.264/AVC and the proposed syntax analysis based error detection. Surprisingly, the proposed syntax analysis based detection method performs better that the packet discard strategy. Since the syntax analysis does not require much complexity, we can conclude that it is beneficial to implement as an alternative to the widely adopted packet discard method.

Acknowledgements

The authors thank mobilkom austria AG for technical and financial support of this work. The views expressed in this paper are those of the authors and do not necessarily reflect the views within mobilkom austria AG.

References

[1] ITU-T Rec. H.264 / ISO/IEC 11496-10, “Advanced Video Coding,” Final Committee Draft, Document JVTE022, Sept. 2002.

[2] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, “Packet switched conversational multimedia applications; Default codecs (Release 6),” ver. 6.4.0, available in `http://www.3gpp.org/`

[3] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, “Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs (Release 6),” ver. 6.8.0, available in `http://www.3gpp.org/`

[4] O. Nemethova, J. Canadas, M. Rupp, “Improved Detection for H.264 Encoded Video Sequences over Mobile Networks,” Proc. of Int. Symp. on Com. Theory and Appl., Ambleside, UK, July 2005.

[5] M. Chen, Y. He, R.L. Lagendijk, “A Fragile Watermark Error Detection Scheme fot Wireless Video Communications,” IEEE Trans. on Multimedia, vol. 7, no. 2, pp. 201-211, April 2005.

[6] C. Weidmann, O. Nemethova, “Improved Sequential Decoding of H.264 Video with VLC Resynchronization,” in Proc. of IST Mobile Summit 2006, Myconos, Greece, June 2006.

[7] M. Barni, F. Bartolini, and P. Bianco, “On the performance of syntax-based error detection in h.263 video coding: A quantitative analysis,” Image and Video Com., Proc. of SPIE, vol.3974, pp.949–956, Jan 2000.

[8] IETF RFC 3828 “The Lightweight User Datagram Protocol (UDP-Lite),” July 2004.

[9] H.264/AVC Software Coordination, “Joint Model Software,” ver.10.2, available in `http://iphome.hhi.de/suehring/tml/`.