To appear in Proc. of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Urbana, Illinois, USA, September 2003

EvalVid - A Framework for Video Transmission and Quality Evaluation

Jirka Klaue, Berthold Rathke, and Adam Wolisz

Technical University of Berlin, Telecommunication Networks Group (TKN)
Sekr. FT5-2, Einsteinufer 25, 10587 Berlin, Germany,
`{jklaue,rathke,wolisz}@ee.tu-berlin.de`

Abstract. With EvalVid[1] we present a complete framework and tool-set for evaluation of the quality of video transmitted over a real or simulated communication network. Besides measuring QoS parameters of the underlying network, like loss rates, delays, and jitter, we support also a subjective video quality evaluation of the received video based on the frame-by-frame PSNR calculation. The tool-set has a modular construction, making it possible to exchange both the network and the codec. We present here its application for MPEG-4 as example. EvalVid is targeted for researchers who want to evaluate their network designs or setups in terms of user perceived video quality. The tool-set is publicly available [11].

1 Introduction

Recently noticeably more and more telecommunication systems are supporting different kinds of real-time transmission, video transmission being one of the most important applications. This increasing deployment causes the quality of the supported video to become a major issue. Surprisingly enough, although an impressive number of papers has been devoted to mechanisms supporting the QoS in different types of networks, much less has been done to support the unified, comparable assessment of the quality really achieved by the individual approaches. In fact, many researchers constrain themselves to prove that the mechanism under study has been able to reduce the packet loss rate, packet delay or packet jitter considering those measures as sufficient to characterize the quality of the resulting video transmission. It is, however, well known that the above mentioned parameters can not be easily, uniquely transformed into a quality of the video transmission: in fact such transformation could be different for every coding scheme, loss concealment scheme and delay/jitter handling.

Publicly available tools for video quality evaluation often assume synchronized frames at the sender and the receiver side, which means they can't calculate the video quality in the case of frame drops or frame decoding errors. Examples are the JNDmetrix-IQ software [4] and the AQUAVIT project [5]. Such tools are not meant for evaluation of incompletely received videos. They are only applicable to videos where every frame could be decoded at the receiver side. Other researchers occupied with video quality

[1] This work has been partially supported by the German research funding agency 'Deutsche Forschungsgemeinschaft' under the program "Adaptability in Heterogeneous Communication Networks with Wireless Access" (AKOM)

evaluation of transmission-distorted video, e.g., [20, 21], did not make their software publicly available. To the best knowledge of the authors there is no free tool-set available which satisfies the above mentioned requirements.

In this paper we introduce EvalVid, a framework and a toolkit for a unified assessment of the quality of video transmission. EvalVid has a modular structure, making it possible to exchange at users discretion both the underlying transmission system as well as the codecs, so it is applicable to any kind of coding scheme, and might be used both in real experimental set-ups and simulation experiments. The tools are implemented in pure ISO-C for maximum portability. All interactions with the network are done via two trace files. So it is very easy to integrate EvalVid in any environments.

The paper is structured as follows: we start with an overview of the whole framework in Section 2, followed by the explanation of the scope of the supported functionality in Section 3 with explanation of the major design decisions. Afterwards the individual tools are described in more detail (Section 4). Exemplary results and a short outline of the usability and further research issues complete the paper.

2 Framework and Design

In Figure 1 the structure of the EvalVid framework is shown. The interactions between the implemented tools and data flows are also symbolized. In Section 3 it is explained **what** can be calculated and Section 4 shows **how** it is done and which results can be obtained.

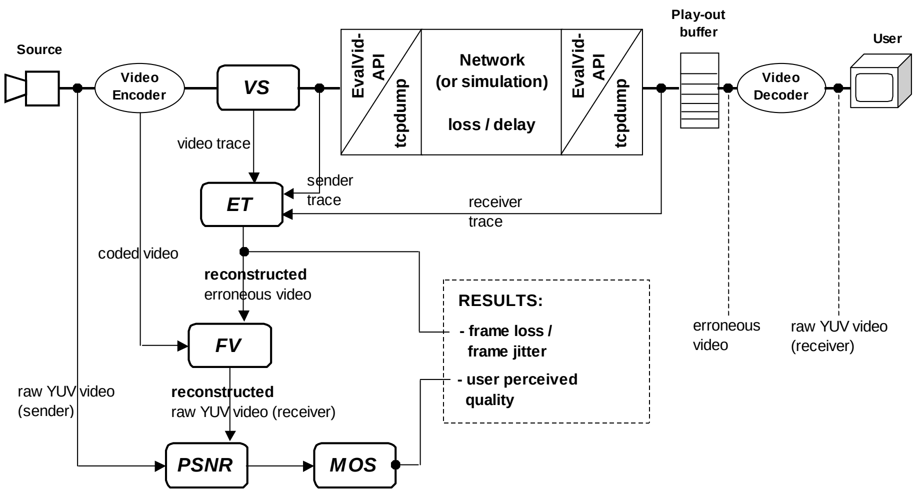


Fig. 1. Scheme of evaluation framework

Also, in Figure 1, a complete transmission of a digital video is symbolized from the recording at the source over the encoding, packetization, transmission over the network, jitter reduction by the play-out buffer, decoding and display for the user. Furthermore the points, where data are tapped from the transmission flow are marked. This information is stored in various files. These files are used to gather the desired results, e.g., loss rates, jitter, and video quality. A lot of information is required to calculate these values.

The required data are (from the sender side):

– raw uncompressed video
– encoded video
– time-stamp and type of every packet sent

and from the receiver side:

– time-stamp and type of every packet received
– reassembled encoded video (possibly errorneous)
– raw uncompressed video to be displayed

The evaluation of these data is done on the sender side, so the informations from the receiver have to be transported back to the sender. Of practical concern is that the raw uncompressed video can be very large, for instance 680 MB for a 3 minute PDA-screen sized video. On the other hand it is possible to reconstruct the video to be displayed from the information available at the sender side. The only additional information required from the receiver side is the file containing the time stamps of every received packet. This is much more convenient than the transmission of the complete (errorneous and decoded) video files from the receiver side.

The processing of the data takes place in 3 stages. The first stage requires the time-stamps from both sides and the packet types. The results of this stage are the frame-type based loss rates and the inter-packet times. Furthermore the errorneous video file from the receiver side is reconstructed using the original encoded video file and the packet loss information. This video can now be decoded yielding the raw video frames which would be displayed to the user. At this point a common problem of video quality evaluation comes up. Video quality metrics always require the comparison of the displayed (possibly distorted) frame with the corresponding original frame. In the case of completely lost frames, the required synchronization can not be kept up (see Section 4.4 for further explanations).

The second stage of the processing provides a solution to this problem. Based on the loss information, frame synchronization is recovered by inserting the last displayed frame for every lost frame. This makes further quality assessment possible. The thus fixed raw video file and the original raw video file are used in the last stage to obtain the video quality.

The boxes in Figure 1 named VS, ET, FV, PSNR and MOS are the programs of which the framework actually consists (see Section 4). Interactions between the tools and the network (which is considered a black box) are based on trace files. These files contain all necessary data. The only file that must be provided from the user of EvalVid is the “receiver trace file”. If the network is a real link, this is achieved with the help of TCP-dump (for details see Section 4, too). If the network is simulated, then this file must be produced by the receiver entity of the simulation. This is explained in the documentation [11].

For the tools within EvalVid only these trace files, the original video file and the decoder are needed. Therefore, in the context of EvalVid the network is just a “black box” which generates delay, loss and possible packet reordering. It can be a real link, such as Ethernet or WLAN, or a simulation or emulation of a network. Since the only interaction of EvalVid and the network is represented by the two trace files (sender and receiver), the network box can be easily replaced, which makes EvalVid very flexible. Similarly, the video codec can also be easily replaced.

3 Supported Functionalities

In this section the parameters calculated by the tools of EvalVid are described, formal definitions and references to deeper discussions of the matter, particularly for video quality assessment, are given.

3.1 Determination of Packet and Frame Loss

Packet loss Packet losses are usually calculated on the basis of packet identifiers. Consequently the network black box has to provide unique packet id’s. This is not a problem for simulations, since unique id’s can be generated fairly easy. In measurements, packet id’s are often taken from IP, which provides a unique packet id. The unique packet id is also used to cancel the effect of reordering. In the context of video transmission it is not only interesting how much packets got lost, but also which kind of data is in the packets. E.g., the MPEG-4 codec defines four different types of frames (I, P, B, S) and also some generic headers. For details see the MPEG-4 Standard [10]. Since it is very important for video transmissions which kind of data gets lost (or not) it is necessary to distinguish between the different kind of packets. Evaluation of packet losses should be done type (frame type, header) dependent. Packet loss is defined in Equation 1. It is expressed in percent.

$$\text{packet loss} \quad PL_T = 100\frac{n_{T_{recv}}}{n_{T_{sent}}} \quad , \qquad \text{where:} \tag{1}$$

T : Type of data in packet (one of all, header, I, P, B, S)
$n_{T_{sent}}$: number of type T packets sent
$n_{T_{recv}}$: number of type T packets received

Frame loss A video frame (actually being a single coded image) can be relatively big. Not only in the case of variable bit rate videos, but also in constant bit rate videos, since the term constant applies to a short time average. I-frames are often considerable larger than the target (short time average) constant bit rate even in “CBR” videos (Figure 2).

It is possible and likely that some or possibly all frames are bigger than the maximum transfer unit (MTU) of the network. This is the maximum packet size supported by the network (e.g. Ethernet = 1500 and 802.11b WLAN = 2312 bytes). These frames has to be segmented into smaller packets to fit the network MTU. This possible segmenting of frames introduces a problem for the calculation of frame losses. In principle

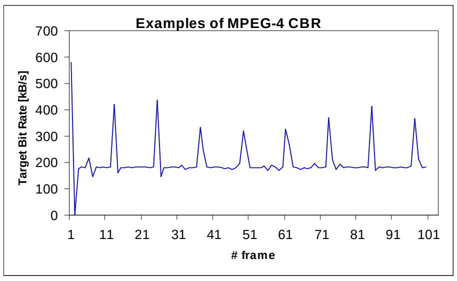


Fig. 2. CBR MPEG-4 video at target bit rate 200 kbps

the frame loss rate can be derived from the packet loss rate (packet always means IP packet here). But this process depends a bit of the capabilities of the actual video decoder in use, because some decoders can process a frame even if some parts are missing and some can't. Furthermore, wether a frame can be decoded depends on which of its packet got lost. If the first packet is missing, the frame can almost never be decoded. Thus, the capabilities of certain decoders has to be taken into account in order to calculate the frame loss rate. It is calculated separately for each frame type.

$$\text{frame loss} \quad FL_T = 100\frac{n_{T_{recv}}}{n_{T_{sent}}} \quad , \qquad \text{where:} \tag{2}$$

$$\begin{aligned} T &: \text{Type of frame (one of all, header, I, P, B, S)} \\ n_{T_{sent}} &: \text{number of type } T \text{ frames sent} \\ n_{T_{recv}} &: \text{number of type } T \text{ frames received} \end{aligned}$$

Determination of Delay and Jitter In video transmission systems not only the actual loss is important for the perceived video quality, but also the delay of frames and the variation of the delay, usually referred to as frame jitter. Digital videos always consists of frames with have to be displayed at a constant rate. Displaying a frame before or after the defined time results in "jerkiness" [20]. This issue is addressed by so called play-out buffers. These buffers have the purpose of absorbing the jitter introduced by network delivery delays. It is obvious that a big enough play-out buffer can compensate any amount of jitter. In extreme case the buffer is as big as the entire video and displaying starts not until the last frame is received. This would eliminate any possible jitter at the cost of a additional delay of the entire transmission time. The other extreme would be a buffer capable of holding exactly one frame. In this case no jitter at all can be eliminated but no additional delay is introduced.

There have been sophisticated techniques developed for optimized play-out buffers dealing with this particular trade-off [17]. These techniques are not within the scope of the described framework. The play-out buffer size is merely a parameter for the evaluation process (Section 4.3). This currently restricts the framework to static play-out buffers. However, because of the integration of play-out buffer strategies into the evaluation process, the additional loss caused by play-out buffer over- or under-runs can be considered.

The formal definition of jitter as used in this paper is given by Equation 3, 4 and 5. It is the variance of the inter-packet or inter-frame time. The “frame time” is determined by the time at which the last segment of a segmented frame is received.

$$\begin{aligned}
\text{inter-packet time} \quad it_{P_0} &= 0 \qquad (3)\\
it_{P_n} &= t_{P_n} - t_{P_{n-1}}\\
\text{where:} \quad t_{P_n} &: \text{ time-stamp of packet number } n\\
\text{inter-frame time} \quad it_{F_0} &= 0\\
it_{F_m} &= t_{F_m} - t_{F_{m-1}}\\
\text{where:} \quad t_{F_m} &: \text{ time-stamp of last segment of frame number } m
\end{aligned}$$

$$\text{packet jitter} \quad j_P = \frac{1}{N}\sum_{i=1}^{N}(it_i - \bar{it_N})^2 \qquad (4)$$

N : number of packets

$\bar{it_N}$: average of inter-packet times

$$\text{frame jitter} \quad j_F = \frac{1}{M}\sum_{i=1}^{M}(it_i - \bar{it_M})^2 \qquad (5)$$

where:

M : number of frames

$\bar{it_M}$: average of inter-frame times

For statistical purposes histograms of the inter-packet and inter-frame times are also calculated by the tools of the framework (see Section 4.3).

3.2 Video Quality Evaluation

Digital video quality measurements must be based on the perceived quality of the actual video being received by the users of the digital video system because the impression of the user is what counts in the end. There are basically two approaches to measure digital video quality, namely subjective quality measures and objective quality measures. Subjective quality metrics always grasp the crucial factor, the impression of the user watching the video while they are extremely costly: highly time consuming, high manpower requirements and special equipment needed. Such objective methods are described in detail by ITU [3, 15], ANSI [18, 19] and MPEG [9]. The human quality impression usually is given on a scale from 5 (best) to 1 (worst) as in Table 1. This scale is called Mean Opinion Score (MOS).

Table 1. ITU-R quality and impairment scale

Scale	Quality	Impairment
5	Excellent	Imperceptible
4	Good	Perceptible, but not annoying
3	Fair	Slightly annoying
2	Poor	Annoying
1	Bad	Very annoying

Many tasks in industry and research require automated methods to evaluate video quality. The expensive and complex subjective tests can often not be afforded. Therefore, objective metrics have been developed to emulate the quality impression of the human visual system (HVS). In [20] there is an exhaustive discussion of various objective metrics and their performance compared to subjective tests.

However, the most widespread method is the calculation of peak signal to noise ratio (PSNR) image by image. It is a derivative of the well-known signal to noise ratio (SNR), which compares the signal energy to the error energy. The PSNR compares the maximum possible signal energy to the noise energy, which has shown to result in a higher correlation with the subjective quality perception than the conventional SNR [6]. Equation 6 is the definition of the PSNR between the luminance component Y of source image S and destination image D.

$$PSNR(n)_{dB} = 20\log_{10}\left(\frac{V_{peak}}{\sqrt{\frac{1}{N_{col}N_{row}}\sum_{i=0}^{N_{col}}\sum_{j=0}^{N_{row}}\left[Y_S(n,i,j) - Y_D(n,i,j)\right]^2}}\right) \quad (6)$$

$$V_{peak} = 2^k - 1$$

$$k = \text{number of bits per pixel (luminance component)}$$

The part under the fraction stroke is nothing but the mean square error (MSE). Thus, the formula for the PSNR can be abbreviated as $PSNR = 20\log\frac{V_{peak}}{MSE}$, see [16]. Since the PSNR is calculated frame by frame it can be inconvenient, when applied to videos consisting of several hundred or thousand frames. Furthermore, people are often interested in the distortion introduced by the network alone. So they want to compare the received (possibly distorted) video with the undistorted[2] video sent. This can be done by comparing the PSNR of the encoded video with the received video frame by frame or comparing their averages and standard deviations.

Another possibility is to calculate the MOS first (see Table 2) and calculate the percentage of frames with a MOS worse than that of the sent (undistorted) video. This method has the advantage of showing clearly the distortion caused by the network at a

[2] Actually, there is always the distortion caused by the encoding process, but this distortion also exists in the received video

glance. In Section 4 you can see an example produced with the MOS tool of EvalVid. Further results gained using EvalVid are shown briefly in Section 5.

Table 2. Possible PSNR to MOS conversion [14]

PSNR [dB]	MOS
> 37	5 (Excellent)
31 - 37	4 (Good)
25 - 31	3 (Fair)
20 - 25	2 (Poor)
< 20	1 (Bad)

4 Tools

This section introduces the tools of the EvalVid framework, describes their purpose and usage and shows examples of the results attained. Furthermore sources of sample video files and codecs are given.

4.1 Files and Data Structures

At first a video source is needed. Raw (uncoded) video files are usually stored in the YUV format, since this is the preferred input format of many available video encoders. Such files can be obtained from different sources, as well as free MPEG-4 codecs. Sample videos can also be obtained from the author.

Once encoded video files (bit streams) exist, trace files are produced out of them. These trace files contain all relevant information for the tools of EvalVid to obtain the results discussed in Section 3. The evaluation tools provide routines to read an write these trace files and use a central data structure containing all the information needed to produce the desired results. The exact format of the trace files, the usage of the routines and the definition of the central data structure are described briefly in the next section and in detail in the documentation [11].

4.2 VS - Video Sender

For MPEG-4 video files, a parser was developed based on the MPEG-4 video standard [10]; simple profile and advanced simple profile are implemented. This makes it possible to read any MPEG-4 video file produced by a conforming encoder. The purpose of VS is to generate a trace file from the encoded video file. Optionally, the video file can be transmitted via UDP (if the investigated system is a network setup). The results produced by VS are two trace files containing information about every frame in the video file and every packet generated for transmission (Table 3 and 4).

Table 3. The relevant data contained in the video trace file is the frame number, the frame type and size and the number of segments in case of (optional) frame segmentation. The time in the last column is only informative when transmitting the video over UDP, so that you can see during transmission, if all runs as expected (The time should reflect the frame rate of the video, e.g. 40 ms at 25 Hz).

Format of video trace file:

Frame Number	Frame Type	Frame Size	Number of UDP-packets	Sender Time
0	H	24	1 segm	40 ms
1	I	9379	10 segm	80 ms
2	P	2549	3 segm	120 ms
3	B	550	1 segm	160 ms
...				

Table 4. The relevant data contained in the sender trace file is the time stamp, the packet id and the packet size. This file is generated separately because it can be obtained by other tools as well (e.g. TCP-dump, see documentation).

Format of sender trace file:

time stamp [s]	packet id	payload size
1029710404.014760	id 48946	udp 24
1029710404.048304	id 48947	udp 1024
1029710404.048376	id 48948	udp 1024
...		

These two trace files together represent a complete video transmission (at the sender side) and contain all informations needed for further evaluations by EvalVid. With VS you can generate these coupled trace files for different video files and with different packet sizes, which can then be fed into the network black box (e.g. simulation). This is done with the help of the input routines and data structures provided by EvalVid, which are described in the documentation. The network then causes delay and possibly loss and re-ordering of packets. At the receiver side another trace, the receiver trace file is generated, either with the help of the output routines of EvalVid, or, in the case of a real transmission, simply by TCP-dump (4.7), which produces trace files compatible with EvalVid.

It is worth noting that although the IP-layer will segment UDP packets exceeding the MTU of underlying layers and will try to reassemble them at the receiving side it is much better to do the segmenting self. If one segment (IP fragment) is missing, the whole packet (UDP) is considered lost. Since it is preferable to get the rest of the segments of the packet I would strongly recommend using the optional MTU segmentation function of VS, if possible.

4.3 ET - Evaluate Traces

The heart of the evaluation framework is a program called ET (evaluate traces). Here the actual calculation of packet and frame losses and delay/jitter takes place. For the calculation of these data only the three trace files are required, since there is all necessary information included (see Section 4.2) to perform the loss and jitter calculation, even frame/packet type based. The calculation of loss is quite easy, considering the availability of unique packet id's. With the help of the video trace file, every packet gets assigned a type. Every packet of this type not included in the receiver trace is counted lost. The type based loss rates are calculated according to Equation 1. Frame losses are calculated by looking for any frame, if one of it's segments (packets) got lost and which one. If the first segment of the frame is among the lost segments, the frame is counted lost. This is because the video decoder cannot decode a frame, which first part is missing. The type-based frame loss is calculated according to Equation 2.

This is a sample output of ET for losses (a video transmission of 4498 frames in 8301 packets).

```
PACKET LOSS                          FRAME LOSS
  H:      1       0     0.0%           H:      1       0     0.0%
  I:   2825       3     0.1%           I:    375       3     0.8%
  P:   2210      45     2.0%           P:   1125      45     4.0%
  B:   3266     166     5.1%           B:   2998     166     5.5%
ALL:   8302     214     2.6%         ALL:   4499     214     4.8%
```

The calculation of inter-packet times is done using Equation 3 and 4). Yet, in the case of packet losses, these formulas can't be applied offhand. This is because in the case of packet losses no time-stamp is available in the receiver trace file for the lost packets. This raises the question how the inter-packet time is calculated, if at least one of two consecutive packets is lost? One possibility would be to set the inter-packet time in the case of the lost packet to an "error" value, e.g., 0. If then a packet is actually received, one could search backwards, until a valid value is found. The inter-packet time in this case would be $t_n - t_{last\_received\_packet}$. This has the disadvantage of not getting a value for every packet and inter-packet times could grow unreasonable big. That's why the approach used by ET is slightly different. If at least one (of the two actually used in every calculation) packets is missing, there will be not generated an invalid value, but rather a value will be "guessed". This is done by calculating a supposable arrival time of a lost packet. We will show how this is done later in this section, and in particular using Equation 7. This practically means that for lost packets the expectancy value of the sender inter-packet time is used. If relatively few packets get lost, this method does not have a significant impact on the jitter statistics. On the other hand, if there are very high loss rates, we recommend another possibility: to calculate only pairwise received packets and count lost packets seperately.

$$\text{arrival time (lost packet)} \quad t_{R_n} = t_{R_{n-1}} + \left(t_{S_n} - t_{S_{n-1}}\right) \qquad (7)$$

where: t_{S_n} : time-stamp of **sent** packet number n

t_{R_n} : time-stamp of **(not) received** packet number n

Now, having a valid time-stamp for every packet, inter-packet (and based on this, inter-frame) delay can be calculated according to Equation 3. Figure 3 shows an example of the inter-frame times calculated by ET.

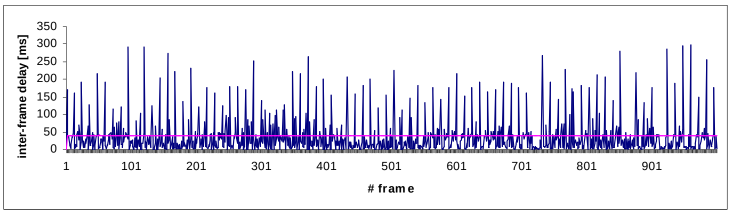


Fig. 3. Example inter-packet times (same video transmission as used for loss calculation)

ET can also take into account the possibility of the existence of certain time bounds. If there is a play-out buffer implemented at the receiving network entity, this buffer will run empty, if no frame arrives for a certain time, the maximum play-out buffer “size”. Objective video quality metrics like PSNR cannot take delay or jitter into account. However, an empty (or full) play-out buffer effectively causes loss (no frame there to be displayed). The maximum play-out buffer size can be used to “convert” delay into loss. With ET you can do this by providing the maximum play-out buffer size as a parameter. The matching of delay to loss is then done as follows:

```
MAX = maximum play-out buffer size

new_arrival_time(0) := orig_arrival_time(0);
FOREACH frame m
  IF (m is lost)
    -> new_arrival_time(m) := new_arrival_time(m-1) + MAX
  ELSE
    IF (inter-frame_time(m) > MAX)
      -> frame is marked lost
      -> new_arrival_time(m) := new_arrival_time(m-1)
                              + MAX
    ELSE
      -> new_arrival_time(m) := new_arrival_time(m-1)
                              + (orig_arrival_time(m)
                              - orig_arrival_tm(m-1));
    END IF
  END IF
END FOREACH
```

Another task ET performs, is the generation of a corrupted (due of losses) video file. This corrupted file is needed later to perform the end-to-end video quality assessment.

Thus another file is needed as input for ET, namely the original encoded video file. In principle the generation of the corrupted video is done by copying the original video packet by packet where lost packets are omitted. One has to pay attention to the actual error handling capabilities of the video decoder in use. It is possible, that the decoder expects special markings in the case of missing data, e.g., special code words or simply an empty (filled with zeros) buffer instead of a missing packet. You must check the documentation of the video codec you want to use.

4.4 FV - Fix Video

Digital video quality assessment is performed frame by frame. That means that you need exactly as many frames at the receiver side as at the sender side. This raises the question how lost frames should be treated if the decoder does not generate “empty” frames for lost frames [3]. The FV tool is only needed if the codec used cannot provide lost frames. How lost frames are handled by FV is described in later in this section. Some explanations of video formats may be required. You can skip these parts if you are already familiar with this.

Raw video formats Digital video is a sequence of images. No matter how this sequence is encoded, if only by exploiting spatial redundancy (like Motion-JPEG, which actually is a sequence of JPEG-encoded images) or by also taking advantage of temporal redundancy (as MPEG or H.263), in the end every video codec generates a sequence of raw images (pixel by pixel) which can then be displayed. Normally such a raw images is just a two-dimensional array of pixels. Each pixel is given by three color values, one for the red, for the green and for the blue component of its color. In video coding however pixels are not given by the three ground colors, but rather as a combination of one luminance and two chrominance values. Both representations can be converted back and forth (Equation 8) and are therefore exactly equivalent.

It has been shown that the human eye is much more sensitive to luminance than to chrominance components of a picture. That’s why in video coding the luminance component is calculated for every pixel, but the two chrominance components are often averaged over four pixels. This halves the amount of data transmitted for every pixel in comparison to the RGB scheme. There are other possibilities of this so called YUV coding, for details see [10].

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= 0.565(B - Y) \\ V &= 0.713(R - Y) \end{aligned} \tag{8}$$

[3] This is a Quality of Implementation issue of the video decoder. Because of the time stamps available in the MPEG stream, a decoder could figure out if one or more frames are missing between two received frames.

$$
\begin{aligned}
R &= Y + 1.403V \\
G &= Y - 0.344U - 0.714V \\
B &= Y + 1.770U
\end{aligned}
$$

The decoding process of most video decoders results in raw video files in the YUV format. The MPEG-4 decoder which I mostly use writes YUV files in the 4:2:0 format.

Decode and display order The MPEG standard basically defines three types of frames, namely I, P and B frames. I frames contain an entire image, which can be decoded independently, only spatial redundancy is exploited. I frames areintra coded frames. P frames are predicted frames; they contain intra coded parts as well as motion vectors which are calculated in dependence on previous (I or P) frames. P frame coding exploits both spatial and temporal redundancy. These frames can only be completely decoded if the previous I or P frame is available. B frames are coded exclusively in dependence on previous and successive (I or P) frames. B frames only exploit temporal redundancy. They can be decoded completely only if the previous and successive I or P frame is available. That's why MPEG reorders the frames before transmission, so that any frame received can be decoded immediately, see Table 5.

Table 5. MPEG decode and display frame ordering

Display order	Frame type	Decode order
1	I	2
2	B	3
3	B	1
4	P	5
5	B	6
6	B	4
...		

Because of this reordering issue, a coded frame does not correspond to the decoded (YUV) frame with the same number. FV fixes this issue, by matching display (YUV) frames to transmission (coded) frames according to Table 5. There are more possible coding schemes than the one shown in this table (e.g. schemes without B frames, with only one B frame in between or with more than two B frames between two I (or P) frames), but the principle of reordering is always the same.

Handling of missing frames Another issue fixed by FV is the possible mismatch of the number of decoded to the original number of frames caused by losses. A mismatch would make quality assessment impossible. A decent decoder can decode every frame, which was partly received. Some decoders refuse to decode parts of frames or to decode B frames, where one of the frames misses from which it was derived. Knowing the handling of missing or corrupted frames by the decoder in use, FV can be tuned to fix the handling weaknesses of the decoder. The fixing always consists of inserting missing frames. There are two possibilities of doing so. The first is to insert an “empty” frame for every not decoded frame (for whatever reason). An empty frame is a frame containing no information. An empty frame will cause certain decoders to display to display a black (or white) picture. This is not a clever approach, because of the usually low differences between two consecutive video frames. So FV uses the second possibility, which is the insertion of the last decoded frame instead of an empty frame in case of a decoder frame loss. This handling has the further advantage of matching the behaviour of a real world video player.

4.5 PSNR - Quality assessment

The PSNR is the base of the quality metric used in the framework to assess the resulting video quality. Considering the preparations from preliminary components of the framework, the calculation of the PSNR itself is now a simple process described by Equation 6. It must be noted, however, that PSNR cannot be calculated if two images are binary equivalent. This is because of the fact that the mean square error is zero in this case and thus, the PSNR couldn’t be calculated according to Equation 6. Usually this is solved by calculating the PSNR between the original raw video file before the encoding process and the received video. This assures that there will be always a difference between to raw images, since all modern video codecs are lossy.

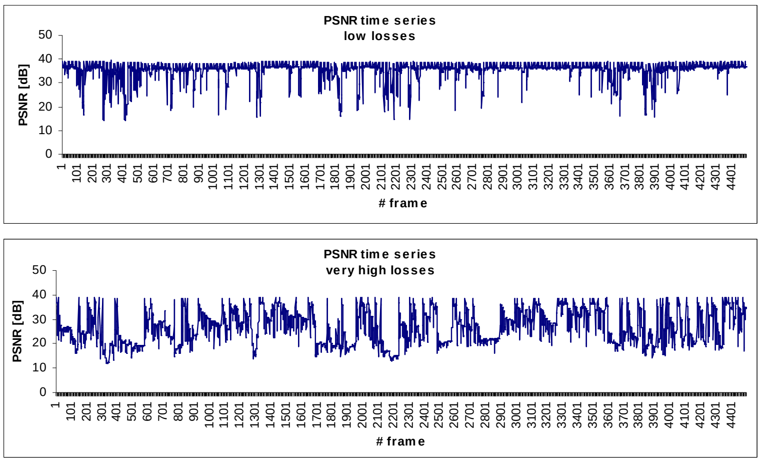


Fig. 4. Example of PSNR (same video transmitted with few and with high losses)

Almost all authors, who use PSNR, only use the luminance component of the video (see Section 4.4). This is not surprising considering the relevance of the Y component for the HVS (Section 3.2). Figure 4 exemplifies two PSNR time series. Other metrics than PSNR can be used, in this case the desired video quality assessment software , e.g., [20], [2] or [4] must replace the PSNR/MOS modules.

4.6 MOS calculation

Since the PSNR time series' are not very concise an additional metric is provided. The PSNR of every single frame is mapped to the MOS scale in Table 1 as described in section 3.2. Now there are only five grades left and every frame of a certain grade is counted. This can be easily compared with the fraction of graded frames from the original video as pictured in Figure 5. The rightmost bar displays the quality of the original video as a reference, "few losses" means an average packet loss rate of 5%, and the leftmost bar shows the video quality of a transmission with a packet loss rate of 25%. Figure 5 pictures the same video transmissions as Figure 4.

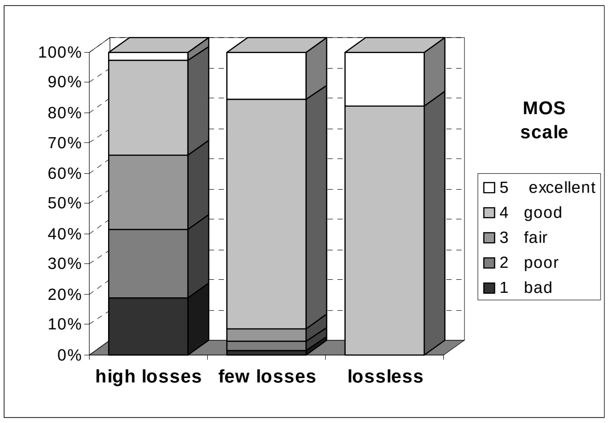


Fig. 5. Example of MOS graded video (same video transmissions as in Figure 4)

The impact of the network is immediately visible and the performance of the network system can be expressed in terms of user perceived quality. Figure 5 shows how near the quality of a certain video transmission comes to the maximal achievable video quality.

4.7 Required 3rd party tools

The programs described above are available as ISO-C source code or pre-compiled binaries for Linux-i386 and Windows. To perform ones own video quality evaluations, you still need some software from other sources. Their integration into the EvalVid framework is described in the documentation. If you want to evaluate video transmission systems using a Unix system or Windows, then you need TCP-dump or win-dump, respectively. You can get them it from:

– `http://www.tcpdump.org`
– `http://windump.polito.it`

You also need raw video files (lossless coded videos) and a video encoder and decoder, capable of decoding corrupted video steams. There are MPEG-4 codecs available from:

– MPEG-4 Industry Forum (`http://www.m4if.org/resources.php`)
– MPEG (`http://mpeg.nist.gov/`)

5 Exemplary Results

This tool-set has been used to evaluate video quality for various simulations [1, 12] and measurements [7]. It proved usable and quite stable. Exemplary results are shown here and described briefly. Figure 6 shows the result of the video quality assessment with EvalVid for a simulation of MPEG-4 video transmission over a wireless link using different scheduling policies and dropping deadlines. The picture shows the percentage of frames with the five MOS ratings, the rightmost bar shows the MOS rating of the original (without network loss) video. It can be clearly seen that the blind scheduling policy does not work very well and that the video quality for the two other policies increases towards the reference with increasing deadlines.

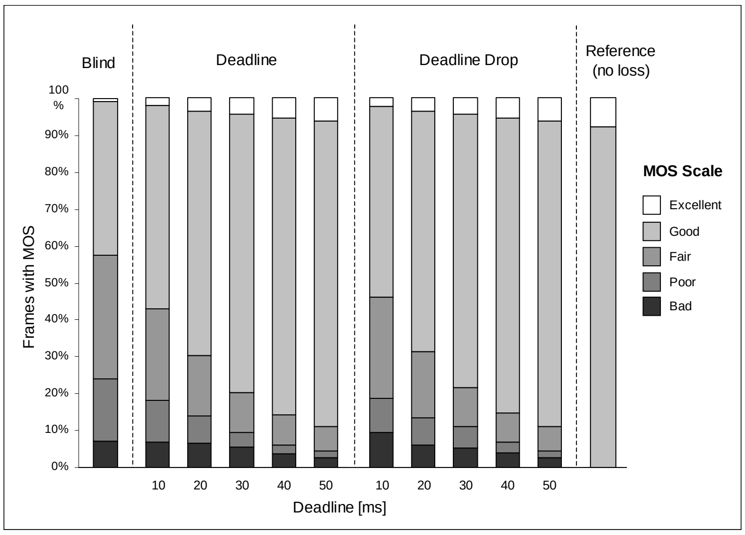
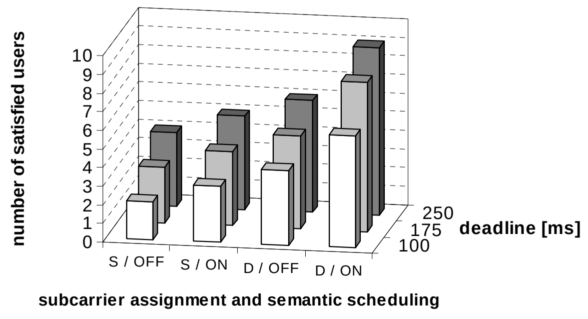


Fig. 6. Example of video quality evaluation (MOS scale) with EvalVid

Similarly, Figure 7 shows the enhancement of user satisfaction with increasing dropping deadlines and better scheduling schemes in a simulation of an OFDM system. The “user satisfaction” was calculated based on the MOS results obtained with EvalVid. The bars in this figure show the number of users that could be supported with a certain mean MOS.



	S / OFF	S / ON	D / OFF	D / ON
□ 100	2	3	4	6
▨ 175	3	4	5	8
■ 250	4	5	6	9

Fig. 7. Example of video quality evaluation (number of satisfied users) with EvalVid

6 Conclusion and Topics to further Research

The EvalVid framework can be used to evaluate the performance of network setups or simulations thereof regarding user perceived application quality. Furthermore the calculation of delay, jitter and loss is implemented. The tool-set currently supports MPEG-4 video streaming applications but it can be easily extended to address other video codecs or even other applications like audio streaming. Certain quirks of common video decoders (omitting lost frames), which make it impossible to calculate the resulting quality, are resolved. A PSNR-based quality metric is introduced which is more convenient especially for longer video sequences than the traditionally used average PSNR. The tool-set has been implemented in ISO-C for maximum portability and is designed modularly in order to be easily extensible with other applications and performance metrics. It was successfully tested with Windows, Linux and Mac OS X.

The tools of the EvalVid framework are continuously extended to support other video codecs as H.263, H.26L and H.264 and to address additional codec functionalities like fine grained scalability (FGS) [13] and intra frame resynchronisation. Furthermore the support of dynamic play-out buffer strategies is subject of future developments. Also it is planned to add support of other applications, e.g. voice over IP (VoIP) [8] and synchronised audio-video streaming. And last but not least other metrics than PSNR-based will be integrated into the EvalVid framework.

Bibliography

[1] A. C. C. Aguiar, C. Hoene, J. Klaue, H. Karl, A. Wolisz, and H. Miesmer. Channel-aware schedulers for voip and mpeg-4 based on channel prediction. to be published at MoMuC, 2003.

[2] Johan Berts and Anders Persson. Objective and subjective quality assessment of compressed digital video sequences. Master's thesis, Chalmers University of Technology, 1998.

[3] ITU-R Recommendation BT.500-10. Methodology for the subjective assessment of the quality of television pictures, March 2000.

[4] Sarnoff Corporation. Jndmetrix-iq software and jnd: A human vision system model for objective picture quality measurements, 2002.

[5] Project P905-PF EURESCOM. Aquavit - assessment of quality for audio-visual signals over internet and umts, 2000.

[6] Lajos Hanzo, Peter J. Cherriman, and Juergen Streit. *Wireless Video Communications*. Digital & Mobile Communications. IEEE Press, 445 Hoes Lane, Piscataway, 2001.

[7] Daniel Hertrich. Mpeg4 video transmission in wireless lans — basic qos support on the data link layer of 802.11b. Minor Thesis, 2002.

[8] H.Sanneck, W.Mohr, L.Le, C.Hoene, and A.Wolisz. Quality of service support for voice over ip over wireless. *Wireless IP and Building the Mobile Internet*, December 2002.

[9] ISO-IEC/JTC1/SC29/WG11. Evaluation methods and procedures for july mpeg-4 tests, 1996.

[10] ISO-IEC/JTC1/SC29/WG11. *ISO/IEC 14496: Information technology - Coding of audio-visual objects*, 2001.

[11] J. Klaue. Evalvid — http://www.tkn.tu-berlin.de/research/evalvid/fw.html.

[12] J. Klaue, J. Gross, H. Karl, and A. Wolisz. Semantic-aware link layer scheduling of mpeg-4 video streams in wireless systems. In *Proc. of Applications and Services in Wireless Networks (AWSN)*, Bern, Switzerland, July 2003.

[13] Weiping Li. Overview of fine granularity scalability in mpeg-4 video standard. *IEEE transaction on circuits and systems for video technology*, March 2001.

[14] Jens-Rainer Ohm. Bildsignalverarbeitung fuer multimedia-systeme. Skript, 1999.

[15] ITU-T Recommendations P.910 P.920 P.930. Subjective video quality assessment methods for multimedia applications, interactive test methods for audiovisual communications, principles of a reference impairment system for video, 1996.

[16] Martyn J. Riley and Iain E. G. Richardson. *Digital Video Communications*. Artech House, 685 Canton Street, Norwood, 1997.

[17] Cormac J. Sreeman, Jyh-Cheng Chen, Prathima Agrawal, and B. Narendran. Delay reduction techniques for playout buffering. *IEEE Transactions on Multimedia*, 2(2):100–112, June 2000.

[18] ANSI T1.801.01/02-1996. Digital transport of video teleconferencing / video telephony signals. ANSI, 1996.

[19] ANSI T1.801.03-1996. Digital transport of one-way video signals - parameters for objective performance assessment. ANSI, 1996.

[20] Stephen Wolf and Margaret Pinson. Video quality measurement techniques. Technical Report 02-392, U.S. Department of Commerce, NTIA, June 2002.

[21] D. Wu, Y. T. Hou, W. Zhu, H.-J. Lee, T. Chiang, Y.-Q. Zhang, and H. J. Chao. On end-to-end architecture for transporting mpeg-4 video over the internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(6):923–941, September 2000.