Quality aware MPEG-2 Stream Adaptation in Resource Constrained Systems

Damir Isović and Gerhard Fohler
Department of Computer Engineering
Mälardalen University, Västerås, Sweden
{damir.isovic,gerhard.fohler}@mdh.se

Abstract

A number of algorithms have been presented for handling software decoding of MPEG-2 streams based on buffering or rate adjustment focusing on providing good average quality. The potentially arising drops in quality are tolerated, e.g., in transmissions over the Internet; they cannot be accepted in high quality consumer products: these mandate real-time methods. When resources, such as processing power or network bandwidth, are limited and not all frames can be handled, best effort decoders incur unnecessary quality decrease while wasting resources.

In this paper, we present a method for quality aware frame selection for MPEG decoding under limited resources, based on realistic timing constraints for the decoding of MPEG streams. Given that not all frames can be processed, it selects those which will provide the best picture quality while matching the available resources, starting only such decoding, which is guaranteed to be completed. We formulate the method as real-time scheduling problem and present its application in an example scheduling algorithm. Results from study based on realistic MPEG-2 video underline the effectiveness of our approach.

1 Introduction

The MPEG-2 standard for video coding [1] is predominant in consumer electronics for DVD players, digital satellite receivers, and TVs today. In such high quality devices, drops in perceived video quality are not tolerated by consumers. Within the next few years, MPEG-2 decoding will move from dedicated hardware to software, for reasons of cost, rapid upgradeability, and configurability. Furthermore, video will not only be watched on classic TV sets, but increasingly displayed on smaller devices ranging from mobile phones to web pads, providing mobility. Consequently, MPEG-2 decoding will be performed in software under limited resources.

Most current software decoders, however, operate under the assumption of sufficient resources, using buffering and rate adjustment based on average-case assumptions. These provide acceptable quality for applications such as video transmissions over the Internet, when decreases in quality, delays, uneven motion or changes in speed are tolerable. In high quality consumer terminals, however, quality losses of such methods are not acceptable. In fact, producers of such devices have argued to mandate the use of hard real-time methods instead [4].

In resource limited situations the processor cannot work fast enough to decode all the frames, the workload for the software decoder has to be reduced. This can be achieved by either by quality reduction, using a downgraded decoding algorithm or frame skipping, i.e., not decoding complete frames. In this paper, we look into frame skipping.

Naive best-effort decoders perform frame skipping by simply running out of time at frame display time, incurring either a sudden disturbance in smoothness, as pictures are missing, or a delay of subsequent frames, disturbing motion speed. As frame decoding starts and proceeds without knowing about timely completion, it may happen that the resources are fully used, but wasted, as partially decoded frames are generally not useful. In extreme cases, the decoding of a large and important frame might just not make it, therefore being lost and impeding quality, while simply skipping to decode a small preceding frame might have freed the resources for completion, with only slight quality reduction. In addition, skipping a frame may affect also other frames due to inter frame dependencies. In a typical movie, a single frame skip can ruin around 0.5 seconds. Thus frame skipping needs appropriate assumptions and constraints about streams [11] to be effective.

A server based algorithm for integrating multimedia and hard real-time tasks has been presented in [2]. It is based on average values for execution times and interarrival intervals. A method for real-time scheduling and admission control of MPEG-2 streams that fits the need for adaptive CPU scheduling has been presented in [6]. The method is not computationally overloaded, qualifies for continuous reprocessing and guarantees QoS. However, no consideration on making priorities on the B frame level has been done. An approach that allows close-to-average-case resource allocation to a single video processing task has been proposed in [18]. It is based on asynchronous, scalable processing, and QoS adaptation. A frame skipping pattern that makes distinction between B frames has been presented in [15]. However, only one skipping criterion, QoS human [14], has been applied when selecting B frames, taking no consideration about frame sizes, buffer and latency requirements, or compression methods used. Most standard real-time schedulers fail to satisfy the demands of MPEG-2 as they do not consider the specifics of this compression standard.

In this paper we present an algorithm for quality aware MPEG-2 stream adaptation in resource constrained systems. It is based on previous work, in which we derived realistic timing constraints for MPEG-2 video decoding without [11] and with [12], and identified a number of misconceptions. The algorithm provides best quality by selecting frames if not all can be decoded under limited resources. It is based on a priority ordering for frame skipping taking frame importance into account. It creates an ensemble of decoding tasks for the frames in Group of Pictures, GOPs, each with timing constraints suited specifically for the particular frame, transforming the GOPs into such tailored for actual demand and available resources. Using a real-time system for resource management, the frame selection algorithm takes into account the actual state of the system, by determining the best GOPs utilizing the available resources and considering the priority ordering for skipping. Thus our algorithm selects frames based on concrete frame knowledge and ensures that only decoding of frames which can be completed in time is started. While the algorithm is independent of the actual guarantee algorithm used, making it suitable to work with a variety of algorithms and paradigms, we present its use with a concrete scheduler. We present results from a study underlining the effectiveness of our approach as compared to standard decoders.

The rest of this paper is organized as follows: in Section 2 we give an overview of the method. Section 3 presents real-time processing model for MPEG playout, and timing constrains for frame decoding. In Section 4 we present a quality aware frame selection algorithm that assign priorities to frames based on some quality criteria, followed by description of the frame skipping algorithm in Section 5. In Section 6 we show the use of the algorithm with a concrete scheduler. The analysis results are reported in Section **??**. Finally, Section 7 concludes the paper.

2 Method overview

The functionality of a MPEG video player is quite simple: it reads a stream of compressed video frames, then decompresses and displays them. Video frames have to be decoded and displayed correctly and in time.

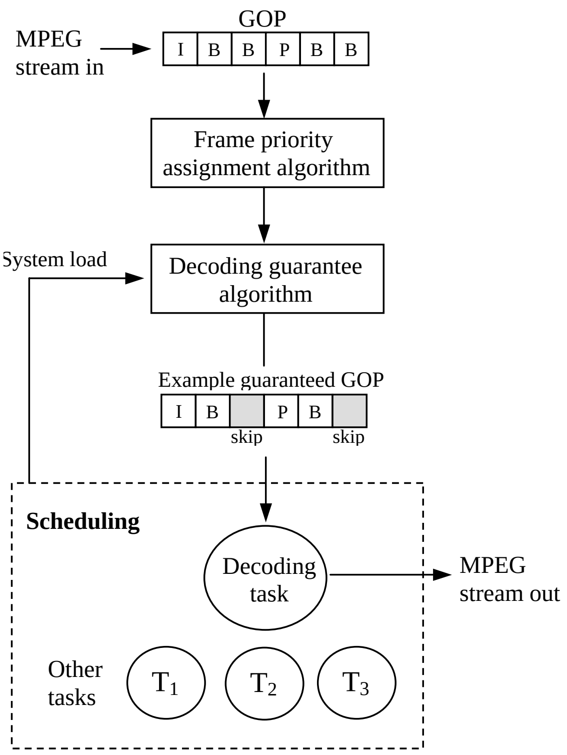


Figure 1. Method overview and system architecture

Figure 1 gives an overview and the system architecture of our approach. We deal with systems with limited resources where frame skipping has to take place. The idea is to assign priorities to each of the frames in a GOP, and then try to guarantee their decoding in time. If not all frames can be decoded in time, then we use the assigned priorities to select which frames to skip first.

We have proposed a number of criteria to be applied when selecting the frames to be decoded. The *frame priority assignment algorithm* uses those criteria to assign importance values to frames. The lower the value for a frame, the sooner the frame will be skipped, compared to the other frames.

The *guarantee algorithm* takes an MPEG stream, the amount of available system resources and frame importance values as input, and produces a best possible MPEG stream. It receives feedback from the system and decides how many and which frames can be decoded in time, depending on the current system load. If the guarantee algorithm fails to ensure decoding of all the frames in the GOP, it will start skipping some of them, starting with the frames that have the least impact on the overall video quality. The output from the guarantee algorithm is the information about which frames can be successfully decoded in time.

The algorithm above needs a mechanism to access the system load online. We use the slot shifting mechanism [8] to offline calculate the amount and the distribution of available resources in the system, and to access this information at runtime. However, other mechanisms can be used as well.

The final result of the stream adaptation process is a tailored MPEG stream that is guaranteed to be decoded and displayed in time. The difference between our method and best-effort based algorithms that randomly skip frames if they run out of time, is that we: a) consider useful only what is finished: partially decoded frames do not contribute to the overall video quality, b) decode only what is guaranteed to finish in time: we will not start decoding a frame unless we can ensure the frame will be completely decoded and displayed in time, and c) select the frames that will give best possible video quality: we use a heuristic to determine which frames in a GOP are more important than the others.

Our approach is applicable on variety of other methods which provide mechanisms for online access of the available system resources. The system resources does not necessarily need to be the available CPU time, it can also be e.g., available network bandwidth: we could apply our method for video streaming through a network, i.e., we take available network bandwidth as input, and create feasible streams which are guaranteed to be transmitted in time.

3 Real-time model for MPEG playout

Here we present real-time processing model for MPEG playout, and timing constrains for frame decoding. As a first step towards the decoding guarantee algorithm, we use those constraints to derive earliest start times and deadlines for the instances of decoding task.

3.1 MPEG video stream structure

The MPEG-2 standard defines three types of frames, I, P and B. The I frames or *intra* frames are coded as still images. They contain absolute picture data and are self-contained, meaning that they require no additional information for decoding. The second kind of frames are P or *predicted* frames. They are forward predicted from the most recently reconstructed I or P frame, i.e., they contain a set of instructions to convert the previous picture into the current one. P frames are not self-contained, i.e., if the previous reference frame is lost, decoding is impossible. The third type is B or *bi-directionally* predicted frames. They use both forward and backward prediction, i.e., a B frame can be decoded from a previous I or P frame, and from a later I or P frame. B frames require resource-intensive compression techniques but they also exhibit the highest compression ratio. No other frames depend on a B frame.

An I frame, together with all of the frames before the next I frame, form a *Group of Pictures (GOP)*. The GOP length is flexible, but 12 or 15 frames is a common value. Furthermore, it is common industrial practice to have a fixed pattern (e.g., $I\ BB\ P\ BB\ P\ BB\ P\ BB$).

B frames are predicted from two I or P frames, one in the past and one in the future. Clearly, information in the future has yet to be transmitted and so is not normally available to the decoder. MPEG gets around the problem by sending frames in the "wrong" order. The frames are sent out of sequence and temporarily stored. For example, original frame sequence $I\ BB\ P$ ... is transmitted as $I\ P\ BB$ ..., so that the future frame is already in the decoder before bi-directional decoding begins. Picture reordering requires additional memory at the encoder and decoder and delay in both of them to put the order right again. The number of bi-directionally coded frames between I and P frames must be restricted to reduce cost and minimize delay.

3.2 Video processing model

In its simplest form, playing out an MPEG video stream requires three activities: input, decoding, and display. These activities are performed by different tasks, separated by input buffer and a set of frame buffers, see figure 2.

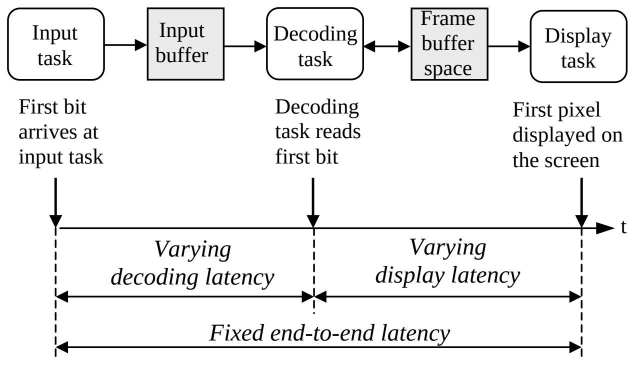


Figure 2. MPEG processing model

The *input task* directly responds to the incoming stream. It places en encoded video stream in the input buffer. In the simple case, the input activity is very regular, and only determined by the fixed bit rate. In a more general case, the input may be of a more bursty character due to an irregular source (e.g. the Internet), or due to a varying multiplex in the transport stream. We assume that the video data is placed in the input buffer at a constant bitrate.

The *decoding task* decodes the input data and puts the decoded frames in the frame buffers. If sufficient buffer space is available, it may work asynchronously, spreading the load more evenly over time. Its deadline is determined by the requirements of the display task. If B frames are present in the stream, the decoder performs frame reordering, i.e., the display order differs from the decoding order. This means that the frames are offered to the display task at irregular intervals. Reference frames are offered to the display task *after* the B frames they helped to decode.

The *display task* is IO bound, and often performed by a dedicated co-processor. It is driven by the refresh rate of the screen. The display task, once started, must always find a frame to be displayed. In the simple case, the display rate equals the frame rate, but we will also consider situations where the display rate is higher than the frame rate.

Once we start to play out an MPEG stream, the *end-to-end latency* is fixed and it is measured from the arrival of the first bit at the input task to the display of the first pixel or line on the screen. If this latency is not fixed, the system cannot work correctly over time. The end-to-end latency is the sum of the *decoding latency*, and the *display latency*, which are not necessarily fixed. If the decoder is asynchronous, i.e. if its activity is determined by the buffer fillings, both latencies can vary, see figure 2.

Since the decoder can be asynchronous, there is a risk of buffer overflow and underflow. Input underflow, and frame buffer overflow occur when the decoder is too fast, i.e., when the decoding latency is too small and/or the display latency too large. The decoder is blocked until the input and/or output task catches up. This can be prevented by synchronization. Input overflow and output underflow occur when the decoder is too slow, i.e., when the decoding latency is too large and/or the display latency is too small. In case of output underflow, the display does not have a new frame to display, but this has been foreseen by retaining the previous frame for display until a new one arrives. Input overflow can be much more serious. In some cases, the input can be delayed, e.g. in case of a DVD player. In other cases, the input task cannot be blocked, especially in case of a broadcast input, where the input buffer must be made large enough to accommodate at least the variation that is allowed by the frame buffers. Theoretically two options are open for the input task: overwrite data at the head of the queue, or drop incoming data. In both cases, reference data will be destroyed, which will lead to a very serious artifact, because the remainder of the GOP cannot be decoded without these reference data. Therefore, preventing overflow at the input is imperative. There are three measures that contribute to preventing overflow: judicious choice of end-to-end latency and input buffer size, speeding up the processing by allocating more processing resources, and preventive load reduction, e.g. by skipping frames, as we do in section 5 of this paper. More detailed description of buffers and latencies can be found in our previous paper [11]

3.3 Timing constraints for MPEG-2 decoding

Video and audio, as well as stream processing in general, have throughput requirements and real-time deadlines. These deadlines are hard in the sense that missing a deadline causes an error, which can render a whole GOP unusable. Timing constraints for an MPEG video decoder stem from roughly three sources: first, *the MPEG stream*, in particular frame ordering and their dependencies, poses mostly relative constraints. Second, *the display rate*, related to the refresh rate of the screen, defines mostly absolute constraints. It depends on hardware characteristics, which in turn define when a picture should be ready to be displayed. Consumer TV sets typically have refresh rates of 50, 60, or 100Hz, computer screens may have more diverse values. Third, the frame buffers incur *resource* and *synchronization* constraints. The number and handling of frame buffers depends on hardware and architecture design, i.e., the constraints will be implementation dependent.

In our previous papers [11, 12] we have derived timing constraints for the decoding task. We proposed *start time constraints*, STC, i.e., the earliest time at which decoding a frame can begin, and *finishing time constraints*, FTC, i.e., the latest time at which decoding a frame has to be completed. For the exact formulation of the timing constraints for MPEG decoding, please refere to our previous work. In this paper, we use those to set the earliest start time and the deadline for the decoder task.

3.4 Decoding task model

As mentioned, an input task (for example a digital video tuner) periodically inserts frames into the input buffer, with a period determined by the frame rate and a display task (for example a video renderer) consumes frames from frame buffers with a period determined by the display rate of the display device. The input task and the display task are synchronized with a fixed end-to-end latency, while the decoding task executes in between asynchronously, since the decoding latency and the display latency can vary.

Start times and deadlines – Decoding task start time and deadline are determined by the requirements of the display task, and the buffer fillings. We use the identified start time and finishing time constaints to set earliest start times and deadlines for frames decoding. Simply, we set the earliest start time and the deadline to be the most strict start time and finishing time constraint. For a frame frame f_i^j, where i is the decoding number of the frame and j is the display number, let $STC^k(f_i^j)$ denote points in time when the corresponding start time constraints are fulfilled, and let $FTC^l(f_i^j)$ represent the same for the finishing time constraints. Then, the earliest start time, est, and the deadline,

dl, for decoding the frame f_i^j are equal to:

$$\begin{aligned} est(f_i^j) &= max\{STC^k(f_i^j)\} \\ dl(f_i^j) &= min\{FTC^k(f_i^j)\} \end{aligned}$$

Execution times – It is difficult to predict worst-case execution times (WCET) for frame decoding. MPEG-2 can use different bitrates which can result in large differences in decoding times for different streams. This could lead to big overestimations of the WCETs. Our analysis [10] shows that the relation between frame size and decoding follows roughly a linear trend. The variations in decoding times for similar frame sizes, however, are significant for the majority of cases, e.g., in the order of 50-100% of the minimum value for B frames. As expected, the frame types exhibit varying decoding time behavior: I frames vary least, since the whole frame is decoded with few options only. On the other hand, B frames, utilizing most compression options, vary most. The frame selection algorithm that we present in section 5 requires known decoding times. There are two ways to make this information available at the guarantee time: offline analysis of the stream which gives exact decoding times, or, in more realistic scenario, prediction at runtime. The focus of this paper is not to provide frame decoding times. Instead, we refer to some previous work. Predicting MPEG execution times has been presented in [3, 5], where the frame decoding time is predicted by frame type and size, and the corresponding predictor is shown to have less than 25% of prediction error.

4 Quality aware frame selection

The latency variation allowed is a design decision, based on the maximum allowed end-to-end latency, and the available buffer space. If the processor cannot work fast enough to meet the time constraints, the decoder has to speed up. There are two ways to do this: quality reduction and frame skipping.

Quality reduction strategies for MPEG decoding and other video algorithms are discussed in [16], [20], [9], and [13]. Control strategies for fine-grained control based on scalable algorithms are proposed in [17] and [19]. These control strategies use a mixture of preventive quality reduction and reactive frame skipping. Quality reduction approach requires algorithms that can be downgraded, with sufficient quality levels to allow smooth degradation.

Frame skips speed up the decoder, and increase the display latency, like a throttle. There are two forms of frame skips, reactive and preventive. A *reactive frame skip* is a frame skip at or after a deadline miss to restore the frame count consistency. In case of a deadline miss, there are two options, aborting the late frame, which is probably almost completely decoded, or completing the late frame, and skipping the decoding of a later frame. The effects of an abortion and of a reactive frame skip on the display latency are shown in [11].

A *preventive frame skip* increases the display latency. Skipping a frame takes a certain time, but much less than decoding it. Instead of rising, which is normal for B frames, the buffer occupancy drops during the frame skipping. The decision to skip preventively is taken at the start of a new frame, and is based on an measurement of the lateness of the decoder.

In this paper, we focus on the preventive frame skipping. Here we present some criteria for quality aware preventive frame selection upon overload situations and a frame skipping algorithm based on these criteria.

4.1 Criteria for preventive frame skipping

Frame skipping needs appropriate assumptions to be effective [11]. Dropping the wrong frame at the wrong time can result in a noticeable disturbance in the played video stream. Here we identify some criteria for frame skipping.

Criterion 1: *Frame type.* According to this criterion, the I frame is the most important one in a GOP since all other frames depend on it. If we lose an I frame, then the decoding of all consecutive frames in the GOP will not be possible. B frames are the least important ones because they are not reference frames. Skipping one B frame will not make any other frame undecodable, while skipping one P frame will cause the loss of all its subsequent frames and the two preceding B frames within the same GOP. If we would apply this criterion only, then we would pull out all B frames first, then P frames and finally the I frame.

Criterion 2: *Frame position in the GOP.* This is applied to P frames. Not all P frames are equally important. Skipping a P frame will cause the loss of *all* its subsequent frames, and the two preceding B frames within the GOP. For instance, skipping the first P frame (P_1) would make it impossible to reconstruct the next P frame (P_2), as well as all B frames that depends on both P_1 and P_2. And if we skip P_2 then we cannot decode P_3 and so on.

Criterion 3: *Frame size.* Applies to B frames. According to the previously presented analysis results, there is a relation between frame size and decoding time, and thus between size and gain in display latency. The bigger the size of the frame we skip, the larger display latency obtained. However, we should not skip unnecessarily big B frames, since they in general contain more picture data.

Criterion 4: *Skipping distribution.* With the same number of skipped B frames, a GOP with *evenly* skipped B frames will be smoother than a GOP with uneven skipped B

frames, e.g if we have a GOP=$IBBPBBPBBPBB$ then even skipping $I-BP-BP-BP-B$ will give smoother video than uneven skipping $I--PBBPBB--$, since the picture information loss will be more spread [14].

Criterion 5: *Buffer size.* Buffer requirements has to be taken into account when designing a frame skipping algorithm. There is no point in having a nice skipping algorithm without having sufficient space to store input data and decoded frames.

Criterion 6: *Latency.* This is not really a criterion, but one must be aware of: an algorithm that takes entire GOP into account requires a large end-to-end latency, and corresponding buffer size.

When deciding the relative importance of frames for the entire GOP, we could assign values to them according to all criteria collectively applied, rather than applying a single criterion. Since the criterion 1 is the strongest one, the I frame will always get the highest priority, as well as the reference frames in the beginning of the GOP, while in some cases we would prefer to skip a P frame towards the end of the GOP than a big B frame close to the GOP start.

4.2 Frame priority assignment algorithm

We apply the skipping criteria above on a GOP and assign different importance values (priorities) to the frames. The lower the value for a frame, the sooner the frame will be skipped, compared to the other frames. The frame priority assignment algorithm goes like this:

1. Apply criterion 1 to assign the highest value to the I frame (equal to the number of frames in the GOP).
2. Apply criterion 2 to assign values to P frames. The closest the distance of a P frame to the start of the GOP, the highest value will be assigned.
3. Apply criterion 3 and 4 to assign values to B frames. Initially set all values for B frames to the lowest P value. Identify all "even-skip" chains for B frames and sort them according to the total bytesize. Decrease the importance values of the B frames, depending on which chain they belong to. The less the total byte of a chain, the less the values are assigned to belonging B frames.
4. Apply criterion 5 and 6 on all frames to adjust priorities if necessary. For example, if there are not enough buffer space for decoding the first P frame, switch its priority with a later P frame.

Example: assume the following GOP with respective bitsizes (taken from a video stream that we analysed):

$$I\ BB\ P\ BB\ P\ BB\ P\ BB =$$

$$\{734136, 89656, 96640, 119368, 89232, 74048, \\ 100680, 32112, 87080, 92064, 18336, 142008\}$$

We want to assign importance values to frames according to our method. The number of frames in the GOP is 12, so the values will be between 1 and 12, 12 being the highest priority. The assigned values after each step are depicted on top of the frames. The frames with values that have changed compared to the previous step will be highlighted by filled style. Also, P and B frames are indexed in order to distinguish between different frames of the same type.

We start by applying criterion 1. According to this criterion, the I frame got the highest value 12, three P frames got the same value 11, and B frames are the least important, with value 10:

12	10	10	11	10	10	11	10	10	11	10	10
I	B_1	B_2	P_1	B_3	B_4	P_2	B_5	B_6	P_3	B_7	B_8

We continue by applying criterion 2 on the P frames. P_1 is closest to the I frame among all P frames. Hence P_1 will keep its assigned value (11), while the values of P_2 and P_3 will get decreased. Since P_2 is closer to the I frame than P_3, it will get higher value than P_3. By this we ensure that in overload situations P_3 will be skipped first, P_2 second and P_1 will be the last one among P frames to skip:

12	10	10	11	10	10	10	10	10	9	10	10
I	B_1	B_2	P_1	B_3	B_4	P_2	B_5	B_6	P_3	B_7	B_8

Since the value of P_3 is now the same as the value of B frames, we even need to decrease the B values to make sure that all P frames will be prioritized before any of the B frames:

12	8	8	11	8	8	10	8	8	9	8	8
I	B_1	B_2	P_1	B_3	B_4	P_2	B_5	B_6	P_3	B_7	B_8

Now we need to assign priorities among B frames. At this point they all have the same priority 8, but we need to assign unique priorities to make sure that we always know which B frames to skip first. We mentioned earlier that the criteria 3 and 4 should not be applied separately. For the criterion 3 we need to compare sizes for B frames. Let $s(f)$ denote the size in bits for a frame f. For the chosen GOP the following holds:

$$s(B_8)>s(B_2)>s(B_1)>s(B_3)>s(B_6)>s(B_4)>s(B_5)>s(B_7)$$

If we apply the frame size criterion alone, then B_{1-8} frames would be assigned values 6, 7, 6, 3, 2, 4, 1 and 8 respectively (B_8 would get the highest value, 8, because it is the largest among all B frames in the GOP). Assume now that we need to skip 4 frames. According to the assigned values, a skipping mechanism that only compares frame sizes would produce the pattern: $I\ BB\ P\ B-\ P\ --\ P-B$, which is not the optimum for the video smoothness, as discussed before. Instead, we need to apply criterion 3 together with criterion 4 to obtain the best possible value assignment with respect to both frame sizes and even distribution of skipped frames. We start by identifying all "even-skip" chains (ESC) of B frames:

$$ESC_1: \quad B_1 \rightarrow B_3 \rightarrow B_5 \rightarrow B_7, \quad sum = 229336\, bits$$
$$ESC_2: \quad B_2 \rightarrow B_4 \rightarrow B_6 \rightarrow B_8, \quad sum = 402238\, bits$$

We compare the total bitsize in both chains, and we assign greatest values to the B frames in the chain with larger size i.e., chain ESC_2. We start by decreasing the values of ESC_1 by the number of frames in ESC_2, i.e., 4; we need those four values for frames in ESC_2. The new assignment:

12	4	8	11	4	8	10	4	8	9	4	8
I	B_1	B_2	P_1	B_3	B_4	P_2	B_5	B_6	P_3	B_7	B_8

Next we do internal value distribution according to the frame sizes, in both ESC_1 and ESC_2. The largest frame in the chain gets the highest value. In ESC_1, B_1 will get value 4 because it is the largest in the chain, and B_7 gets the smallest value 1. Similarly, in ESC_2, B_8 keeps the value 8 and B_2 gets the lowest value, 4. The final value assignment:

12	4	7	11	3	5	10	2	6	9	1	8
I	B_1	B_2	P_1	B_3	B_4	P_2	B_5	B_6	P_3	B_7	B_8

By doing this kind of value assignments for B frames we find the compromise between even skipping and frame sizes, because we make skipping decision based not only on the frame size but also on the relation to the other B frames in the GOP. i.e., the influence on the entire GOP. Finally, a frame skipping algorithm based on frame importance values would skip frames in the following order:

													(GOP size)
1)	I	B	B	P	B	B	P	B	B	P	B	B	1677822
2)	I	B	B	P	B	B	P	B	B	P	$-$	B	1659486
3)	I	B	B	P	B	B	P	$-$	B	P	$-$	B	1627374
4)	I	B	B	P	$-$	B	P	$-$	B	P	$-$	B	1538142
5)	I	$-$	B	P	$-$	B	P	$-$	B	P	$-$	B	1448486
6)	I	$-$	B	P	$-$	$-$	P	$-$	B	P	$-$	B	1374438

...and so on...

5 Online stream adaptation

In the previous section we showed a method to assign priorities to the frames within a GOP. Here we present how we can online adapt video streams by running a guarantee algorithm for frame decoding. It takes an MPEG stream, the amount of available CPU bandwidth and frame priorities, and produces a tailored MPEG stream, as discussed in section 2.

5.1 Guarantee algorithm for frame decoding

Many available software decoders perform badly in the case of a deadline miss; they simply skip the current frame, without taking any consideration about the frame importance. In the worst case, the current frame could be an I frame, which would ruin the entire GOP. We base our skipping decision on the assigned importance values between frames. If the current frame is an important one, we do not skip it, instead we skip a less important frame.

For each frame in a GOP, we check how much available resources do we have between the earliest start time and the deadline for the frame decoding. If the amount of available resources is less than the decoding execution demand of the frame, we must skip frames. When a frame is skipped, timing constraints for other frames in the GOP can be relaxed.

Assume, for example, we need to skip frame f_3 in a GOP. If we skip framef_3, then the required display times (and hence the decoding deadlines) of all preceding frames can be relaxed, since we are neither decoding nor displaying f_3. We shift deadlines of all preceding frames to the right, i.e., $dl(f_2)$ becomes equal to $dl(f_3)$ and $dl(f_1)$ becomes $dl(f_2)$. Similarly, the earliest start times of the successor frames are shifted to the left, since the frames will become available in the input buffer earlier if we skip the current frame. An approach similar to this has been proposed by Wüst et al. in [18], which relaxes the decoding deadlines of the remaining frames upon a deadline miss of the decoder task. .

5.2 Pseudo-code

Let GOP_c denote currently guaranteed group of pictures, and let f_s be the frame that is to be skipped, i.e., the one with currently lowest importance value among remaining frames in GOP_c. Also let $\mathcal{P}$ and $\mathcal{S}$ denote subsets of GOP_c, containing predecessor and successor frames of f_s respectively:

$$\begin{aligned}
GOP_c &= \{\overbrace{f_1, f_2, ..., }^{\mathcal{P}} f_s, \overbrace{f_{s+1}, f_{s+2}, ..., f_n}^{\mathcal{S}}\} \\
\mathcal{P} &= \{\mathcal{P} \subseteq GOP_c \mid \forall f_j \in \mathcal{P},\ 1 < j < s\} \\
\mathcal{S} &= \{\mathcal{S} \subseteq GOP_c \mid \forall f_k \in \mathcal{S},\ s < k < n\}
\end{aligned}$$

Pseudo code (see the comments below in parallel):

```
1:  $\forall f_i \in GOP_c$
2:      while $availableResources[est(f_i), dl(f_i)] \leq c(f_i)$
```

```
3:        $f_s = minImportanceValue(GOP_c)$
4:        $GOP_c = GOP_c - f_s$
5:        $\forall f_j \in \mathcal{P}$
              $dl(f_j) = dl(f_{j+1})$
          $\forall f_k \in \mathcal{S}$
              $est(f_k) = est(f_{k-1})$
```

Comments:

1. Go through the current GOP frame by frame.
2. Compare the execution demand of the current frame with the available resource between the earliest start time and the deadline of the frame.
3. Get the frame with the currently minimum importance value.
4. Remove skipped frame from the current GOP.
5. Relax deadlines for the predecessor frames and start times of the successor frames. The current frame f_i belongs either to $\mathcal{P}$ or $\mathcal{S}$, i.e., either its deadline or start time constraint is relaxed, which means in the next step of the while loop the amount of available resources for f_j will be bigger compared to the previous step (before skipping f_s).

Complexity - For each frame that we skip (in the while-loop), the amount of frames in GOP_c will decrease (the for-loop), giving the worst-case complexity $O(N^2)$, where N is the number of frames in the GOP. Considering very small values for N (in most cases 12-15 frames), the algorithm is relatively cost-efficient to run online.

6 Evaluation

As an example, we show how we can adjust streams in the context of our previous work, i.e., combined offline and online scheduling. We get the amount of available CPU resources by using the slot shifting mechanism [8], and apply our guarantee algorithm to create a feasible stream.

6.1 Available system resources

The frame guarantee algorithm needs a mechanism to access the amount and the distribution of available resources online. We use the slot shifting mechanism [8] to offline calculate the amount and the distribution of available resources in the system. First, an offline scheduler [7] creates scheduling tables for the selected periodic tasks with complex constraints. It allocates tasks to nodes and resolves complex constraints by constructing sequences of task executions. The resulting offline schedule consist of independent tasks with start times and deadlines, which can be re-scheduled by EDF at runtime, preserving original constraints. Second, the offline schedule is divided into a set of *disjoint execution intervals*. Offline scheduled task scan be executed flexibly within their intervals, i.e., they can be "shifted" in order to accommodate for tasks that arrive at runtime, i.e., aperiodic and sporadic tasks. Third, we need to know amount and location of resources available after offline tasks are guaranteed. *Spare capacities* to represent available resources are then calculated for each interval. The spare capacities, sc, of an interval I_i are calculated as:

$$sc(I_i) \; = \; |I_i| - \sum_{T \in I_i} wcet(T) + min(sc(I_{i+1}), 0) \quad (1)$$

where T are the tasks that belong to I_i. The length of I_i, minus the sum of the activities assigned to it, is the amount of idle time in that interval. These have to be decreased by the amount "lent" to subsequent intervals: Tasks may execute in intervals prior to the one they are assigned to. Then they "borrow" spare capacity from the "earlier" interval. The reader is referred to [8] for details about the calculation of intervals and spare capacities.

6.2 Online access of available system resources

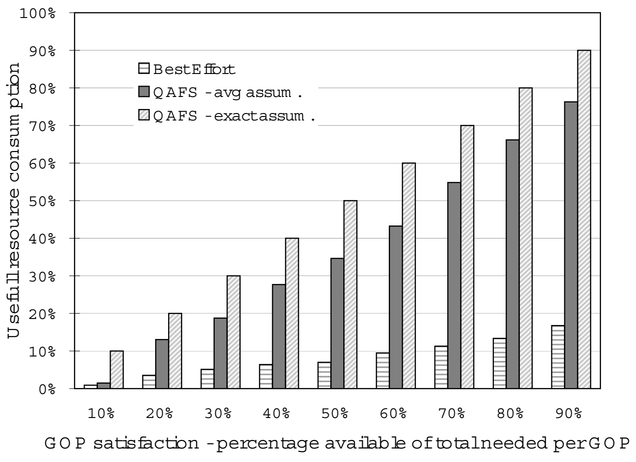
At runtime, we can access the amount and the distribution of available resources via intervals and spare capacities. For example, at any point in time at runtime we can easily calculate the amount of spare capacity between two time slots, t_1 and t_2, by simply summing up the spare capacities of the intervals between them. Let I_{start} denote the interval that contains $t1$ and I_{end} the one containing t_2. Then, the total amount of spare capacities between t_1 and t_2 it is equal to the remaining spare capacity I_{start} , plus the sum of spare capacities of all full intervals between I_{start} and i_{end}, plus the remaining spare capacity of I_{end}:

$$sc[t_1, t_2] = sc_r(I_{start}) + \sum_{I_i \in (t1, t2)} sc(I_i) + sc_r(I_{end}) \quad (2)$$

This mechanism suits perfectly well for online stream adaptation, providing a simple and efficient way to access the amount of available CPU time at runtime, but as mentioned above, other mechanisms can be used as well.

We have implemented and analyzed the quality aware frame selection algorithm (*QAFS*) and compared our algorithm with a naive, best-effort approach.

A naive algorithm will try to decode even those frames that cannot be decoded in time. It will start to decode a frame, and when the decoding deadline miss occurs, it will simply throw it away, unnecessarily wasting the CPU time. Furthermore, if no frame distinction between frame types is done, a best-effort algorithm could, in the worst case, ruin entire GOP by skipping the I frame.



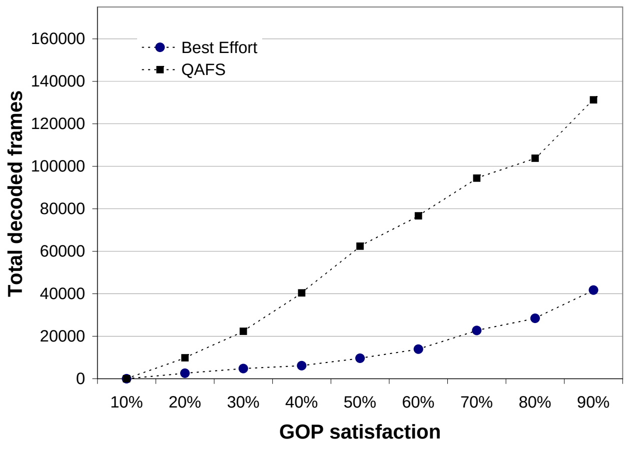


Figure 3. Simulation results - Quality Aware vs Best Effort decoding

Our algorithm selects frames based on concrete frame knowledge and ensures that only decoding of frames which can be completed in time is started. It will not start decoding a frame unless we can ensure the frame will be completely decoded and displayed in time. Besides, *QAFS* selects the frames that will give best possible video quality, according to the frame selection criterion discussed in subsection 4.1.

We have compared *QAFS* and best effort with respect to the useful resource consumption per GOP and the total number of decoded frames. Each value in both graphs of figure 3 is derived from running the simulations on about 15000 GOPs. The system load per GOP is randomly distributed between the frames.

Useful resource consumption - Here we compared the time per GOP spent on useful decoding, i.e., fully decoded frames that contribute to the overall picture quality (as picture data and/or reference data). Wasted decoding time is the one spent on partial decoding frames that must be aborted due to decoding deadline misses.

The *x*-axis in the figure represents the GOP satisfaction degree, which is the ratio between the resources needed for timely decoding of a GOP and the available system resource given to the GOP by the stream. E.g., if GOP satisfaction is 30%, then the GOP is given 30% resources of what it needs for decoding of all its frames. The *y*-axis shows how much of the granted time is spent on the useful decoding.

We have simulated the case with known exact execution times for the decoding of frames, that we measured offline, and with the average decoding times for the respective frame types. As expected, the analysis shows that $QAFS$ will not waste any resources at all if the exact execution times, or worst-case execution times, are known upon guaranteeing. Although there are not yet completely accurate method to predict decoding times online, we have showed the efficiency of our algorithm once when such method is available. In a more realistic case with the average decoding times, $QAFS$ will waste some resources due to the fact that it will guarantee decoding for some frames that cannot be decoded in time, since it performs frame guarantee based on the average decoding times and not the exact decoding times. So, whenever it accepts a frame that has the exact execution time that is larger than the average execution time, some resources will be wasted. Still, it performs much better than the naive, best-effort algorithm, as it can be seen from the figure. Since the GOP load is distributed between all the frames in the GOP, the best effort algorithm will miss decoding deadlines whenever the total load in the execution window of the currently decoded frame is larger than the execution demand of the frame. This means that all decoding spent on the frame upon the deadline miss is wasted. On the other hand, frames are skipped by*QAFS*, it will adjust the start times and the deadlines of the remaining frames in the GOP, giving them higher probability to meet their deadlines.

Total decoded frames - Here we compare the total number of frames successfully decoded by respective algorithm. As expected, the best-effort approach performs worse because it makes no distinction between frames: when a P frame is skipped, all its referring frames will also be skipped, and if the I frame is skipped, then no other frame in the GOP can be decoded. Our algorithm will never skip any reference frames unless it is absolutely necessary since it skips frames based on assigned importance values.

7 Conclusions

In this paper, we presented a quality aware frame selection algorithm for software decoding of MPEG-2 streams under limited resources, based on realistic timing constraints for MPEG decoding. The algorithm selects frames providing best video quality if not all can be completed due to limited resources. It is based on an priority ordering for frame dropping taking frame importance into account. The algorithm creates an ensemble of decoding tasks for the frames in the Groups of Pictures in the stream, each with parameters suited specifically for the particular frame, instead of working with fixed, constant task parameters for periodic tasks. Applying real-time guarantee tests, the algorithm determines the best set of frames to utilize the resources available and providing video quality. While the frame selection algorithm is independent of the actual scheduling algorithm used, we presented an examplatory scheduler for our frame selection method.

We are currently studying the application of our algorithm for networks with limited bandwidth. In particular in mobile, wireless situations, bandwidth will vary, adding another degree of variability to the problem. We are also investigating the sub frame level, e.g., slices and macroblocks for quality trade-offs under limited resources. To underline the independence of the frame selection algorithm, an application with server algorithms is underway. Furthermore, we are looking into how we can offline guarantee minimum quality of service for MPEG streams with online adaptation.

8 Acknowledgements

The authors wish to express their gratitude to Liesbeth Steffens and Clemens Wüst for fruitfull discussions and to the reviewers for helpful comments.

References

[1] ISO/IEC 13818-2: Information technology - generic coding of moving pictures and associated audio information, part2: Video. 1996.

[2] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

[3] A. Bavier, A. Montz, and L. Peterson. Predicting MPEG execution times. In *Proceedings of ACM International Conference on Surement and Modeling of Computer Systems (SIGMETRICS 98)*, Madison, Wisconsin, USA, June 1998.

[4] R. J. Bril, M. Gabrani, C. Hentschel, G. C. van Loo, and E. F. M. Steffens. Qos for consumer terminals and its support for product families. In *Proceedings of the International Conference on Media Futures*, Florence, Italy, May 2001.

[5] L. O. Burchard and P. Altenbernd. Estimating decoding times of MPEG-2 video streams. In *Proceedings of International Conference on Image Processing (ICIP 00)*, Vancouver, Canada, September 2000.

[6] M. Ditze and P. Altenbernd. Method for real-time scheduling and admission control of MPEG-2 streams. In *The 7th Australasian Conference on Parallel and Real-Time Systems (PART2000)*, Sydney, Australia, November 2000.

[7] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Austria, Apr. 1994.

[8] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proc. 16th Real-time Systems Symposium*, Pisa, Italy, 1995.

[9] C. Hentschel, R. Braspenning, and M. Gabrani. Scalable algorithms for media processing. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, Thessaloniki, Greece, pp. 342-345, October 2001.

[10] D. Isovic and G. Fohler. Analysis of MPEG-2 streams. Technical Report at Malardalen Real-Time Research Centre,Vasteras, Sweden, March 2002.

[11] D. Isovic, G. Fohler, and L. F. Steffens. Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, June 2003.

[12] D. Isovic, G. Fohler, and L. F. Steffens. Real-time issues of MPEG-2 playout in resource constrained systems. *Journal of Embedded Computing (JEC), special issue 3*, June 2004, Pre-copy available upon request.

[13] Y. C. John Tse-Hua Lan and Z. Zhong. MPEG2 decoding complexity regulation for a media processor. In *Proceedings of the 4th IEEE Workshop on Multimedia Signal Processing (MMSP)*, Cannes, France, pp. 193 - 198, October 2001.

[14] J. K. Ng, K. R. Leung, W. Wong, V. C. Lee, and C. K. Hui. Quality of service for MPEG video in human perspective. In *Proceedings of the 8th Conference on Real-Time Computing Systems and Applications (RTCSA 2002)*, Tokyo, Japan, March 2002.

[15] J. K.-Y. Ng, C. K.-C. Hui, and W. Wong. A multi-server design for a distributed MPEG video system with streaming support and QoS control. In *Proceedings of the 7th International Conference on Real-Time Systems and Applications (RTCSA'00)*, Cheju Island, South Korea, December 2000.

[16] S. Peng. Complexity scalable video decoding via idct data pruning. In *Digest of Technical Papers IEEE International Conference on Consumer Electronics (ICCE)*, pp. 74-75, June 2001.

[17] C. Wüst. Quality level control for scalable media processing applications having fixed CPU budgets. In *Proceedings Philips Workshop on Scheduling and Resource Management (SCHARM01)*, 2001.

[18] C. Wüst, L. Steffens, R. J. Bril, and W. F. Verhaegh. Qos control strategies for high-quality video processing. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Catania, Sicily, Italy, June 2004.

[19] C. Wüst and W. Verhaegh. Dynamic control of scalable meadia applications. In *Algorithms in Ambient Intelligence*. editors: E.H.L. Aarts, J.M. Korst, and W.F.J. Verhaegh, Kluwer Academic Publishers, 2003.

[20] Z. Zhong and Y. Chen. Scaling in MPEG-2 decoding loop with mixed processing. In *Digest of Technical Papers IEEE International Conference on Consumer Electronics (ICCE)*, pp. 76-77, June 2001.