Real-time issues of MPEG-2 playout in resource constrained systems

Damir Isović, *Member, IEEE,* Gerhard Fohler, *Member, IEEE,* Liesbeth Steffens, *Member, IEEE*

***Abstract*— Decoding MPEG-2 video streams imposes hard real-time constraints for consumer electronic devices such as TV sets. The freedom of encoding choices provided by the MPEG-2 standard results in high variability inside streams, in particular with respect to frame structures and their sizes.**

In this paper, we discuss real-time issues for MPEG-2 playout. We present results from a study of realistic MPEG-2 video streams to analyze the validity of common assumptions for software decoding and identify a number of misconceptions. We identify constraints imposed by frame buffer handling and discuss their implications on decoding architecture and timing. Finally, we identify realistic timing constraints demanded by high quality MPEG-2 software video decoding.

***Index Terms*— MPEG-2, quality of service, real-time video processing, timing constraints, misconceptions about MPEG, resource constrained systems**

I. Introduction

THE Moving Picture Experts Group (MPEG) standard for coded representation of digital audio and video [1], is used in a wide range of applications. In particular MPEG-2 has become the coding standard for digital video streams in consumer content and devices, such as DVD movies and digital television set top boxes for Digital Video Broadcasting (DVB). MPEG encoding has to meet diverse demands, depending, e.g., on the medium of distribution, such as overall size in the case of DVD, maximum bitrate for DVB, or encoding speed for live broadcasts. In the case of DVD and DVB, sophisticated provisions to apply spatial and temporal compression are applied, while a very simple, but quickly coded stream will be used for the live broadcast. Consequently, video streams, and in particular their decoding demands will vary greatly between different media.

The encoded content has to be decoded and played out. Decoding can be performed in hardware or in software, or in a mix of both. Both dedicated and programmable decoders can be based on average-case requirements if they provide means to gracefully handle overload situations. If not, both must support worst-case requirements. However, in a software implementation, it is possible to use the slack on the processor for other applications in average case. With dedicated hardware, there are no such possibilities. As a consequence, the behavior of a software decoder will be less regular than that of a dedicated hardware decoder. Coping with these irregularities is one of the objectives dealt with in this article.

While in the simplest case of sufficient resources, MPEG decoding is straight forward, i.e., simply a matter of transmitting and decoding to display frames with the required frequency, the considerable variations in the streams render such approaches too costly for many cases. If the processor cannot work fast enough to decode all the frames, the decoder has to speed up. There are two ways to do this: quality reduction, and frame skipping. With the quality reduction strategy, the decoder reduces the load by using a downgraded decoding algorithm, while frame skipping means that not all frames are decoded and displayed, i.e., some of the frames are skipped. In this paper, we focus on the frame skipping approach. Frame skipping can be used sparingly to compensate for sporadic high loads, or it can be used frequently if the load is structurally too high.

Many algorithms for software decoding of MPEG video streams use buffering and rate adjustment based on average-case assumptions. These provide acceptable quality for applications such as video transmissions over the Internet, when drops in quality, delays, uneven motion or changes in speed are tolerable. However, in high quality consumer terminals, such as home TVs, 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 [2]. A server based algorithm for integrating multimedia and hard real-time tasks has been presented in [3]. 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 [4]. The method is not computationally overloaded, qualifies for continuous re-processing and guarantees QoS. However, no consideration on making priorities on the B frame level has been done.

It is difficult to predict WCET for decoding parts. 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. Work on predicting MPEG execution times has been presented in [5], [6]. Most standard real-time schedulers fail to satisfy the demands of MPEG-2 as they do not consider the specifics of this compression standard.

This paper is based on our previous work [7]. We derive realistic timing constraints for MPEG-2 video decoding. We analyze realistic MPEG streams and match the results with common assumptions about MPEG, identifying a number of misconceptions. The correct assumptions are needed to identify realistic timing constraints for MPEG processing. Even frame skipping needs appropriate assumptions to be effective. Dropping the wrong frame at the wrong time can result in a noticeable disturbance in the played video stream. We discuss frame buffer handling and its impact on decoding design and temporal requirements. Based on correct assumptions, we provide guidelines for real-time MPEG processing, such as choosing buffer sizes and latency to derive the appropriate timing constraints. These constraints call for novel scheduling algorithms to appropriately meet the exact constraints without quality loss due to misconceptions about the stream characteristics.

The rest of this paper is organized as follows: We begin by giving an overview of MPEG-2 video and decoding in section II, followed by the analysis of realistic video streams in section III. We describe buffer handling in section IV and give requirements for end-to-end flow control in section VI. The results are combined into the derivation of timing constraints in section VII. Finally, section VIII concludes the paper.

II. Playing MPEG streams

In this section we present the main characteristics of MPEG-2 video stream and give an overview how the stream is processed, i.e., buffering, decoding and displaying. A complete description of the MPEG compression scheme is beyond the scope of this paper. For details on MPEG see e.g., [1], [8], [9]. The text presented in this section is summarized in figure 1.

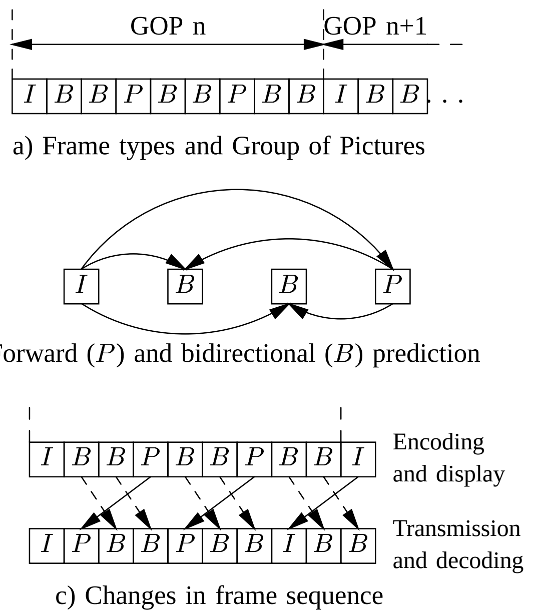

GOP n
GOP n+1
I B B P B B P B B I B B . . .
a) Frame types and Group of Pictures
I B B P
b) Forward (P) and bidirectional (B) prediction
I B B P B B P B B I
Encoding and display
I P B B P B B I B B
Transmission and decoding
c) Changes in frame sequence

Fig. 1. MPEG-2 video stream

A. Frame types

The MPEG-2 standard defines three types of frames, I, P and B. The I frames or *intra* frames are simply frames coded as still images. They contain absolute picture data and are self-contained, meaning that they require no additional information for decoding. I frames have only spatial redundancy providing the least compression among all frame types. Therefore they are not transmitted more frequently than necessary.

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. On average, P frames require roughly half the data of an I frame, but our analysis also showed that this is not the case for a significant number of cases.

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. They contain vectors describing where in an earlier or later pictures data should be taken from. They also contain transformation coefficients that provide the correction. B frames are never predicted from each other, only from I or P frames. As a consequence, no other frames depend on B frames. B frames require resource-intensive compression techniques but they also exhibit the highest compression ratio, on average typically requiring one quarter of the data of an I picture. Our analysis showed that this does not hold for a significant number of cases.

B. Group of Pictures

Predictive coding, i.e., the current frame is predicted from the previous one, cannot be used indefinitely, as it is prone to error propagation. A further problem is that it becomes impossible to decode the transmission if reception begins part-way through. In real video signals, cuts or edits can be present across which there is little redundancy. In the absence of redundancy over a cut, there is nothing to be done but to send from time to time a new reference picture information in absolute form, i.e., an I frame. As I decoding needs no previous frame, decoding can begin at I coded information, for example, allowing the viewer to switch channels. 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., $I\,BB\,P\,BB\,P\,BB\,P\,BB$). However, more advanced encoders will attempt to optimize the placement of the three frame types according to local sequence characteristics in the context of more global characteristics. Note that the last B frame in a GOP requires the I frame in the next GOP for decoding and so the GOPs are not truly independent. Independence can be obtained by creating a *closed* GOP which may contain B frames but ends with a P frame.

C. Transmission order

As mentioned above, 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. Figure 1-c shows that although the original frame sequence is $I\ BB\ P$ ..., this 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, if delay is an issue.

D. MPEG-2 video processing

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

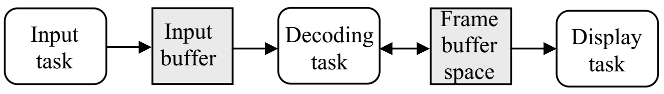


Fig. 2. MPEG tasks and buffers

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.

III. Analysis of realistic MPEG streams

In this section we present an analysis of MPEG-2 video streams taken from original DVDs. We have analyzed 12 diverse streams and matched our results with the common MPEG assumptions. Due to space limitations we report only representative results for selected DVD movies. The complete results for all analyzed movies can be found in [10].

A. The analysis

We have measured frame sizes, decoding execution times, and GOP statistics such as total GOP sizes, the number of open and closed GOPs, the number of GOPs where the I frame is not the largest one, I,B,P frame patterns etc. Then we matched the obtained results with some common assumptions about MPEG video stream.

Since some video contents are more sensitive for quality reduction than others [11], we have analyzed different types of movies; action movies, dramas, and cartoons, see table I. Column GOP in the table I represents the GOP structure of the streams, i.e., it refer to the length and distance between reference frames respectivelly, e.g. GOP strucure (12,3) means I-to-I distance is 12, while I-to-P and P-to-P distance is 3.

TABLE I
Some representative MPEG streams

Genre	Length	Fps	Resolution	Mbit/s	GOP
Action movie	118 min	25	720x576	9800	(12,3)
Drama	107 min	25	720x576	8700	(12,3)
Cartoon	104 min	25	720x576	6000	(12,3)

B. Simulation environment

The MPEG video streams have been extracted from original DVD movies. To extract the data out of an MPEG video stream, we have implemented a C-program. The decoding execution time measurements were performed on several PC computers, with different CPU speed (in the range 0.5-2.0 GHz). The time for measuring decoding execution times was equivalent to the length of the movies.

C. Analysis results

GOP and frame size statistics of the selected movies are presented in table II. We have also analyzed the relations between frame sizes on the individual GOP basis, see table III. “GOP with same length 82%” in the table III means that in the analysed movie 82% of the GOP had the same length, e.g. 12 frames per GOP, while 18% of the GOPs did not follow that pattern i.e., contains less or more frames. “9%” in the column “$P > I$” of the table III means that in 9% of the GOPs there are at least one P frame that is larger then the I frame. The other colums in the tables are quite self-explanatory. Furthermore, we have measured the decoding times for different frame types, see figure 3.

D. Common assumptions about MPEG

Here we present some common assumptions about MPEG and match them with our analysis results. We have looked into stream assumptions (1-4), frame size assumptions (5-8), and a decoding time assumption (9).

Assumption 1: *The sequence structure of all GOPs in the same video stream is fixed to a specific I,P,B frame pattern.* This is not true. For example, in 18% of the GOPs in the action movie the GOP length was not 12 frames. Not all GOPs consist of the same fixed number of P and B frames following the I frame in a fixed pattern. That is because more advanced encoders will attempt to optimize the placement of the three picture types according to local sequence characteristics in the context of more global characteristics.

Assumption 2: *MPEG streams always contain B frames.* Not true. We have been able to identify MPEG streams that contain

TABLE II
FRAME SIZE STATISTICS FOR SELECTED ANALYZED MPEG STREAMS (IN BYTES)

Genre	Avg I:P:B size ratio	Nr of frames	I frames			P frames			B frames		
			min	max	average	min	max	average	min	max	average
action	4:2:1	179412	11	247073	63263	2	152000	29352	4	96131	18525
drama	6:3:2	173054	17	183721	58985	4	126229	28893	4	79552	19054
cartoon	6:2:1	121406	7178	140152	84318	159	137167	31943	159	111405	14398

TABLE III
GOP STATISTICS

Genre	Open GOPs	Closed GOPs	Standard GOP length	Number of GOPs where					
				I largest	P largest	B largest	$P > I$	$B > I$	$B > P$
action	83%	17%	82%	90%	9%	1%	9%	5%	39%
drama	98%	2%	92%	94%	5%	1%	6%	3%	37%
cartoon	99%	1%	98%	92%	7%	1%	8%	1%	12%

only I and P frames (IPP), or even only the I frames in some rare cases. I frame only is an older MPEG-2 technology that does not take advantage of MPEG-2 compression techniques. The IPP technology provides high quality digital video and storage, making it suitable for professional video editing. B frames provide the highest compression ratio, making the MPEG file smaller and hence more suitable for video streaming, but if the file size is not an issue, they can be excluded from the stream.

Assumption 3: *All B frames are coded as bi-directional.* This is not true. There are B frames that do have bi-directional references, but in which the majority of the macroblocks are I blocks. If the encoder cannot find a sufficiently similar block in the reference frames, it simply creates an I block.

Assumption 4: *All P frames contribute equally to the GOP reconstruction.* Not true. The closer the P frame is to the start of the GOP, the more other frames depend on it. For example, without the first P frame in the GOP, P_1, it would be impossible to decode the next P frame, P_2, as well as all the B frames that depends on both P_1 and P_2. In other words, P_2 depends on P_1, while the opposite is not the case.

Assumption 5: *I frames are the largest and B frames are the smallest.* It holds on average. In all the movies that we analyzed, the average sizes of the I frames were larger than the average sizes of the P frames, and P frames were larger than B frames on average. However, our analysis showed that this assumption is not valid for a significant number of cases. For example, in the action movie we have a case with 9% GOPs in which P have the largest size, and 1% of GOPs where a B frame is the largest one (see table III), which corresponds roughly to 8 and 1 minutes respectively in a 90 minute film. Such deviations from average cannot be ignored.

Assumption 6: *An I frame is always the largest one in a GOP*. This is not true. For example in the action movie the I frame was not the largest in 12% of the cases (in 9% of the cases some P frame was larger than the I frame, and in 3% of the GOPs, a B frame was larger than the I frame).

Assumption 7: *B frames are always the smallest ones in a GOP.* Not true. For example, in the drama movie, a B frame was larger than the I frame in 3% of the cases, and larger than a P frame in 37% of the cases. As a consequence, even the assumption that P frames are always larger than B frames is also not valid. As another example, we found a GOP where the B frame is almost 100 times larger than the I frame ($B \approx 1MB, I \approx 12kB$).

Assumption 8: Assumption 8: - *I,P and B frame sizes vary with minor deviations from the average value of I,P and B.* Not true. In the action movie, B frame sizes vary greatly around an average of 18525 bytes. The interval between 0.5 and 1.5 of average holds only some 60% of frames.

Assumption 9: *Decoding time depends on the frame size and it is linear.* While some results on execution times for special kinds of frames have been presented, e.g., [6], a (linear) relationship between frame size and decoding time cannot be assumed in general. Our analysis 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 (see figure 3): 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.

IV. LATENCY AND BUFFER REQUIREMENTS

The input, decoding and display tasks are separated by buffers: one input buffer used for storing the input video bit-stream data, and a frame buffer space that contains at least two frame buffers. In this section we describe system latency and buffer requirements.

A. Latency

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, see figure 4.

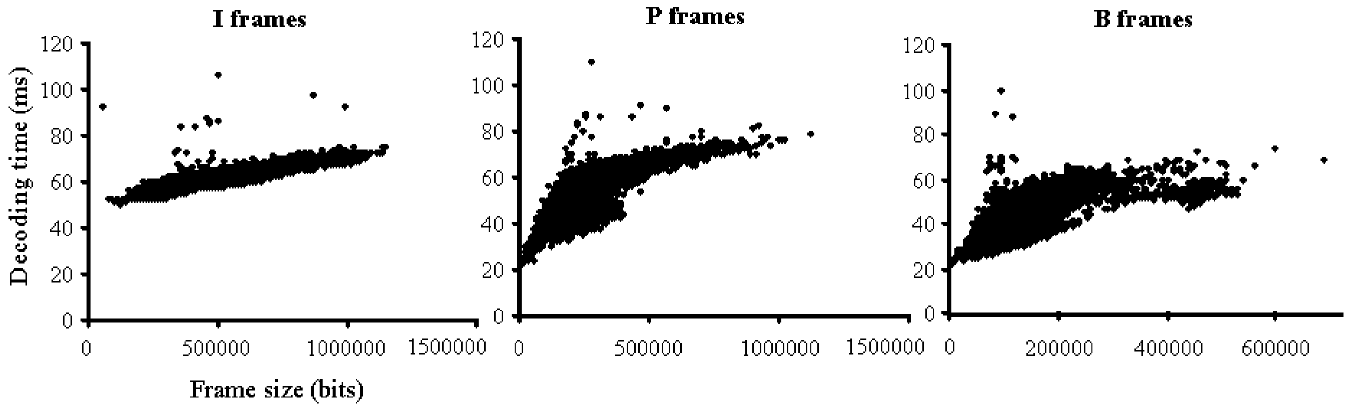


Fig. 3. Decoding execution times as a function of frame bitsize

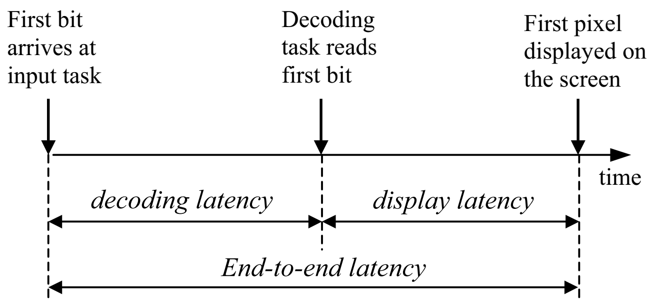

First bit arrives at input task
Decoding task reads first bit
First pixel displayed on the screen
time
decoding latency
display latency
End-to-end latency

Fig. 4. End-to-end latency for MPEG playout

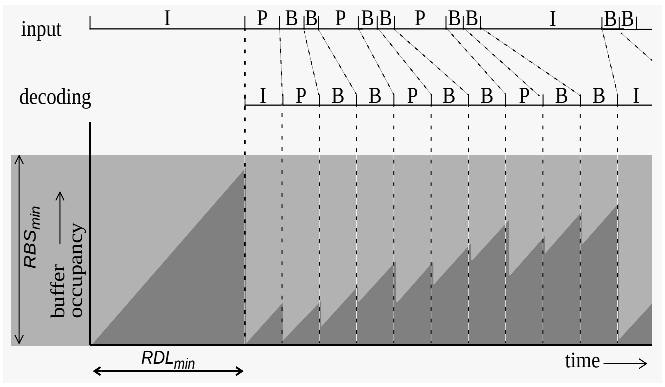


Fig. 5. Reference decoder - minimum decoding latency

The initial decoding latency is measured from the arrival of the first bit at the input task to the reading of the first bit of the first frame, after the header, by the decoder. The initial display latency is measured from the reading of the first bit of the first frame, after the header, by the decoder, to the display of the first pixel on the screen. If the decoding task is strictly periodic, the decoding and display latencies are constant. If the decoder is asynchronous, i.e. if its activity is determined by the buffer fillings, both latencies can vary.

B. Input buffer requirements

The input buffer serves several purposes. First, it has to compensate for the irregular data size. This irregularity is bounded, and the bounding is encoded in the stream, in the form of a parameter called *VBVbuffer size*, see MPEG video standard [1]. VBV stands for Video Buffering Verifier, a hypothetical decoder that starts when the first frame has completely arrived in its input buffer, and retrieves a complete encoded frame out of the input buffer at the start of a new frame period. The contents of the VBV input buffer never exceeds VBV buffer size. Figure 5 depicts the time lines and the buffer occupancy for a reference decoder that corresponds to the VBV. It shows minimum decoding latency, RDL_{min}, and minimum buffer size, RBS_{min} at the start of a new stream. The time lines represent the input and decoding tasks, respectively. Because of the fixed bit rate, BR, the duration of inputting one picture is directly proportional to the number of bits this picture takes up in the encoded stream. The buffer occupancy rises linearly during the decoding of each frame, and drops vertically at the start of a new frame, when the picture data are removed from the input buffer. The buffer occupancy is zero when the first picture has just been removed from the input buffer.

Second, the input buffer has to compensate for varying decoding times, which are not foreseen by the encoder. Therefore, this compensation cannot be bounded a priori.

Third, a realistic decoder retrieves the data from the input buffer according to its processing. The resulting non-zero retrieval time relaxes the buffer requirement, but can also not be bounded a priori. Therefore, the input buffer size is essentially a design choice, closely related to the initial decoding latency and the desired end-to-end latency. Once the size of the input buffer is chosen, the maximum decoding latency (RDL_{max}) is fixed:

$$RDL_{max} = \frac{IBS}{BR}$$

where IBS is the input buffer size, and BR the bit rate.

C. Frame buffers requirements

The frame buffers serve a dual purpose. They serve as reference buffers for the decoder and as input buffers for the display task, or output buffer for the decoding task. It is possible that a certain frame buffer is used in both capacities at the same time. This makes frame buffer management somewhat more complicated than input buffer management.

The display task cannot start until the first frame has been placed in the output buffer, and does not release the current output buffer until a second output buffer is available (double buffering scheme). In this way, the display task always has a frame to display. If the stream contains two or more B frames in sequence, the minimum number of frame buffers needed is 4: two for the reference frames, one for the B frame being displayed, one for the B frame being decoded.

The use of four frame buffers allows a certain irregularity in the delivery of output frames by the decoder. Figures 6 and 7 depict the behavior of a regular reference decoder, which takes exactly one frame period to decode a frame. In the first period in figure 6, a new I frame is being decoded in frame buffer $FB1$. This I frame is needed to decode the B frames B_7 and B_8 (that belong to the previos GOP but are being transmitted after the I frame which is their backward reference frame). In the next period, B_7 can be decoded, and in the third period, B_7 can be displayed, while B_8 is being decoded. If B_7 is the n-th frame to be displayed, it is the $(n+1)$-th frame to be decoded. Therefore, the minimum display latency equals *two* frame periods. If there are no B frames, there is no frame reordering, and the minimum display latency will be *one* frame period instead of two. In figure 6, the decoding cannot be done

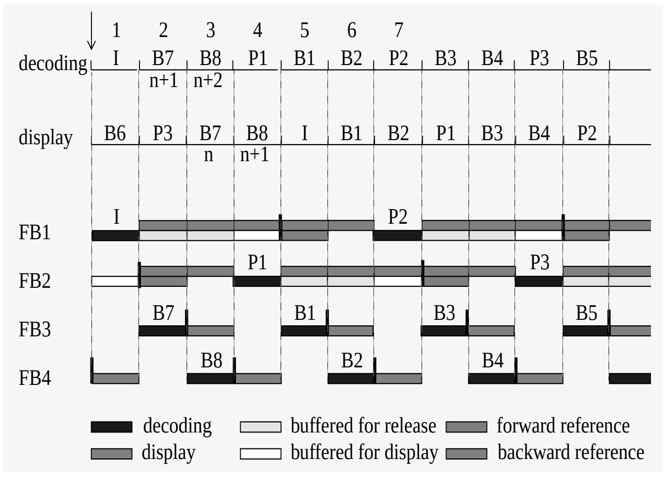


Fig. 6. Reference decoder - minimum display latency

with less than four frame buffers, but these four frame buffers do allow a larger display latency. Figure 7 depicts a situation in which the display latency is maximised. The B frames are displayed not when they are completely decoded, but when the buffer is needed to decode the next frame. Now the n-th frame is being displayed while the $(n+3)$-th frame is being decoded, i.e. the display latency equals three frame periods. Thus the display latency is bounded between the minimum of two frame periods and a maximum of three frame periods.

D. Buffer overflow and underflow

Since the decoder is 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

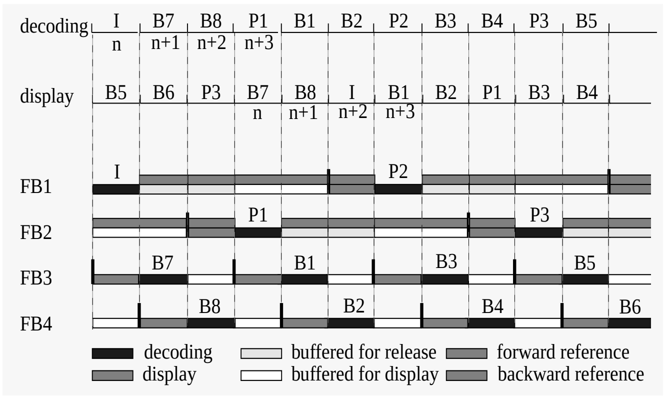
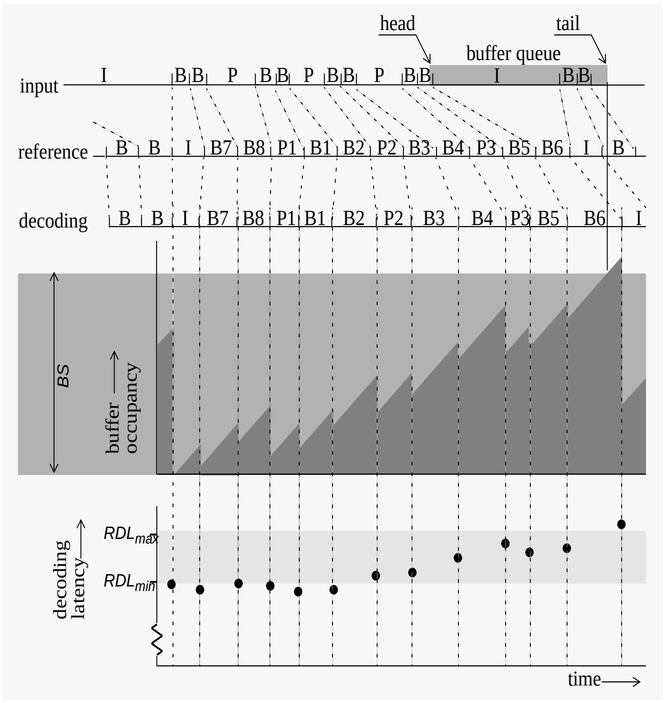


Fig. 7. Reference decoder - maximum display latency

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. Figure 8 depicts such an overflow situation.



head
tail
buffer queue
input
I B B P B B P B B P B B I B B
reference
B B I B7 B8 P1 B1 B2 P2 B3 B4 P3 B5 B6 I B
decoding
B B I B7 B8 P1 B1 B2 P2 B3 B4 P3 B5 B6 I
BS
buffer occupancy
decoding latency
RDLmax
RDLmin
time

Fig. 8. Input buffer owerflow

The overflow is most likely to occur close to the end of a GOP. At that time, the buffer will be mainly filled with I frame data. Since input time is proportional to input size, the shaded rectangle in the input time-line can be interpreted as the buffer contents when the overflow occurs. The decoder reads from the head of the buffer queue, the input task writes to the tail of the buffer queue. At the head of the queue there will be some B frame data (from B6, the B frame that is being decoded when the overflow occurs), which have not yet been removed. At the tail of the queue the input data may reach up to the next P frame.

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. This will be discussed in details in the next section.

V. END-TO-END FLOW CONTROL

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. Whichever strategy is chosen, we assume that the system organisation is such that the display task is never without data to display. This is not difficult to achieve. If a decoded frame does not arrive on time, and the display task has to redisplay the previous frame, this is a deadline miss for the decoder. With the given arrangement deadline misses have a penalty, in the form of a perceived quality reduction. Moreover, since the frame count has to remain consistent, the decoder must skip one frame.

A. Quality reduction

With the quality reduction strategy, the decoder reduces the load by using a downgraded decoding algorithm. Quality reduction for MPEG decoding and other video algorithms is discussed in [12], [13], [14], and [15]. This approach has two advantages over frame skipping. In general the decoding load is higher when there is more motion, but in that case, skipping frames may be more visible than reducing the quality of individual pictures. Moreover, quality reduction can be more subtle, whereas skipping frames is rather coarse grained. Control strategies for fine-grained control based on scalable algorithms are proposed in [16] and [17]. These control strategies use a mixture of preventive quality reduction and reactive frame skipping. The main disadvantage of the quality reduction approach is that it requires algorithms that can be downgraded, with sufficient quality levels to allow smooth degradation. Such algorithms are not yet widely available.

B. Frame skipping

Frame skips speed up the decoder, and increase the display latency, like a throttle. Unfortunately, the corrective step is rather coarse grained: the display latency is increased by a complete frame period. If the range of allowable display latencies is not large enough, this may lead to oscillation, in which frame skips and bounces on frame buffer overflow both are very frequent.

Frame skipping does not come for free. At the very least, the start of the new frame has to be found and the intermediate data have to be thrown away. 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 figures 9 and 10. In the former case, the display latency stays low, and a next deadline miss is to be expected soon. In the latter case, the display latency is drastically reduced, because the decoder will be blocked due to output buffer overflow. An additional frame buffer would give more freedom, and a more stable system, at the cost of using additional memory. In both cases, we have to make sure that the input buffer is large enough to allow the minimal display latency.

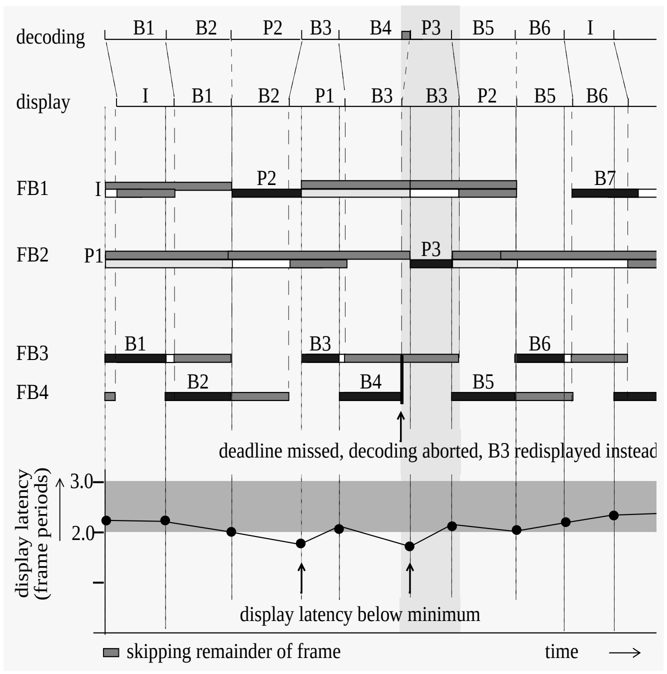

decoding
B1
B2
P2
B3
B4
P3
B5
B6
I
display
I
B1
B2
P1
B3
B3
P2
B5
B6
FB1
I
P2
B7
FB2
P1
P3
FB3
B1
B3
B6
FB4
B2
B4
B5
deadline missed, decoding aborted, B3 redisplayed instead
display latency (frame periods)
3.0
2.0
display latency below minimum
skipping remainder of frame
time

Fig. 9. Deadline missed - frame aborted

A *preventive frame skip* preventively 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. The effect of a preventive frame skip on the display latency is depicted in figure 11, where the decoding of $B3$ is skipped (1). To keep up the appropriate frame count, $P1$

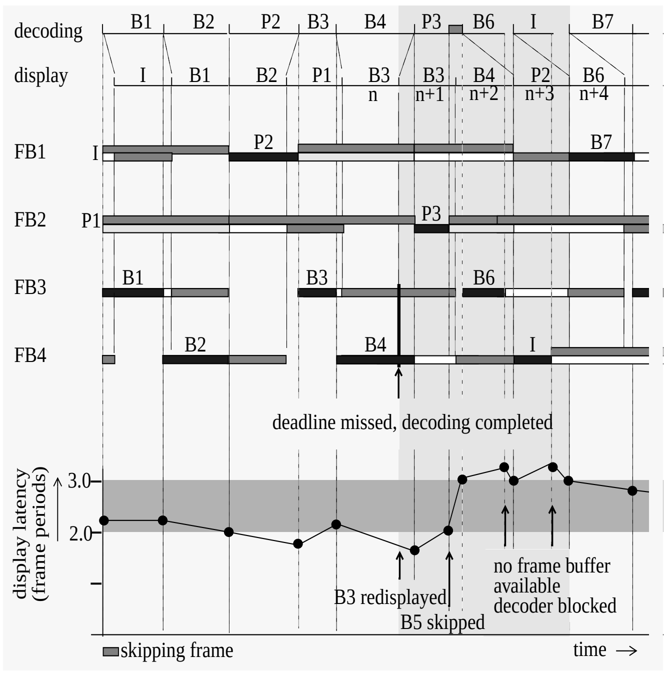

decoding
B1 B2 P2 B3 B4 P3 B6 I B7
display
I B1 B2 P1 B3 B3 B4 P2 B6
n n+1 n+2 n+3 n+4
FB1
FB2
FB3
FB4
deadline missed, decoding completed
display latency (frame periods)
3.0
2.0
B3 redisplayed
B5 skipped
no frame buffer available decoder blocked
skipping frame
time →

Fig. 10. Deadline missed - subsequent frame skipped

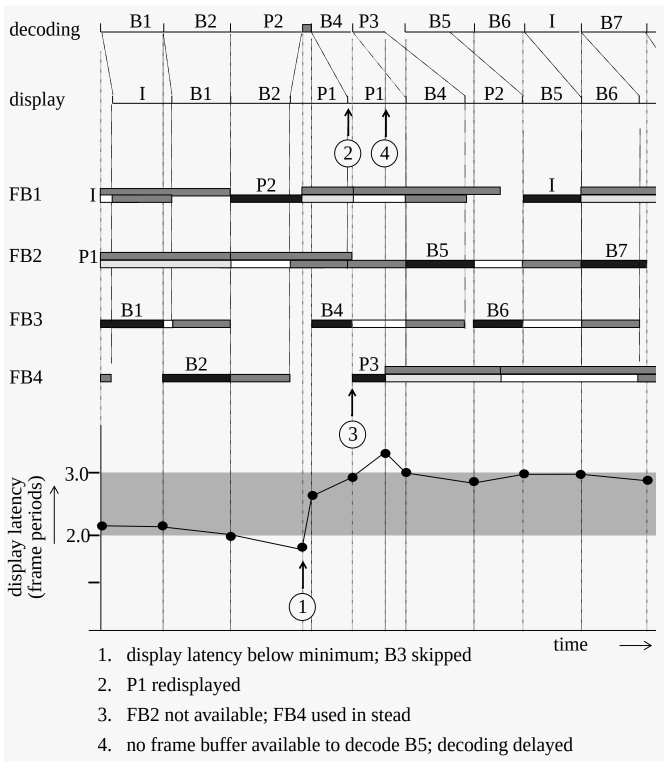

decoding
B1 B2 P2 B4 P3 B5 B6 I B7
display
I B1 B2 P1 P1 B4 P2 B5 B6
FB1
FB2
FB3
FB4
display latency (frame periods)
3.0
2.0
time →
1. display latency below minimum; B3 skipped
2. P1 redisplayed
3. FB2 not available; FB4 used in stead
4. no frame buffer available to decode B5; decoding delayed

Fig. 11. Preventive frame skipping

is displayed twice (2). Because of this, $FB2$ is not available decode the next reference frame $P3$. Therefore, $P3$ has to be decoded in $FB4$ (3). The buffer organization must be such that this is allowed. When $B5$ is to be decoded, the decoder must wait for a free reference frame buffer (4). Because of the frame skip, and the short decoding time for $B4$, the display latency rises to more than 3 frame periods at the start of $P3$. The display latency could even rise up to 4 frame periods at the start of $B5$ if the decoding of $B4$ and $P3$ would be very fast.

In the next section we present some criteria for quality aware frame selection upon overload situations.

VI. Quality aware frame selection

As mentioned in the previous section, one way of speeding up decoding upon overload situations, is to skip some frames. However, frame skipping needs appropriate assumptions to be effective. Dropping the wrong frame at the wrong time can result in a noticeable disturbance in the played video stream. In this section we identify some criteria for frame skipping and propose an algorithm for quality aware frame selection when it is not possible to decode all frames in time.

A. Criteria for preventive frame skipping

Not all the frames are equally important for the overall video quality. Dropping some of them will result in more degradation than others. Here we identify some criteria to decide the relative importance of frames.

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 purpose of skipping is to increase display latency. So, the bigger the size of the frame we skip, the larger display latency obtained.

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 [11].

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.

B. Frame selection algorithm

Here is an example of a frame selection algorithm that makes skipping decisions based on some of the criteria above. We apply the skipping criteria on a GOP and assign different importance values to the frames. The lower the value for a frame, the sooner the frame will be skipped. The number of importance values is equal to the number of frames in the GOP. That will provide for unique priorities between frames. Note that, even if we need to check entire GOP to assign values to the frames, we do not need to buffer the entire GOP, since we only do a look-ahead in the stream where we check the GOP structure and count frame sizes.

Here is the pseudo-code for the assignment of the importance values among frames in a GOP:

```
Let:
    $N$          = GOP length
    $M$          = distance between reference frames
    $\mathcal{P}$          = a set containing all $P$ frames in the GOP
    $\mathcal{B}$          = a set containing all $B$ frames in the GOP
    $v(f)$       = importance value of frame $f$
    $ESC_i$      = $i^{th}$ even-skip chain of $B$ frames

        /* apply criterion 1 on the all frames */
1:      $v(I) = N$

2:      $\forall P_i \in \mathcal{P}, 1 \leq i \leq |\mathcal{P}|$
            $v(P_i) = N - i$

3:      $\forall B_k \in \mathcal{B}, 1 \leq k \leq |\mathcal{B}|$
            $v(B_k) = min[v(P_i) \mid 1 \leq i \leq |\mathcal{P}|] - 1$

        /* apply criterion 3 and 4 on B frames */
4:      $ESC_1 = \{B_1\} \cup \{B_{1+j*M} \mid 1 \leq j \leq \frac{N}{M-1}\}$
        $\forall i, 2 \leq i \leq |\mathcal{B}^*|$
            $ESC_i = \{B_i\} \cup \{B_{i+j*M} \mid 1 \leq j \leq \frac{N}{M-1}\}$
            $if \;\; sum(ESC_i) > sum(ESC_{i-1})$
                $swap(ESC_i, ESC_{i-1})$
            $\forall B_k \in ESC_i$
                $v(B_k) = v(B_k) - |ES_{i-1}|$
```

Comments:

1) Assign the highest value to the I frame (equal to the number of frames in the GOP)
2) The set $\mathcal{P}$ contains all P frames, $\mathcal{P} = \{P_1, P_2, ..., P_k\}$, sorted according to their position in GOP (P_1 is closest to the I frame, i.e, the first P frame in the GOP, while P_k is the last frame in the GOP). The longer the distance from the I frame, the lower the importance value (P_1 will get the highest value and P_k the lowest one).
3) Initially set all values for B frames to the lowest P value (e.g. P_k above).
4) 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 valuest are assigned to belonging B frames.

C. Example

Assume the following GOP with respective bitsizes (taken from the action movie):

$$I \;\, BB \;\, P \;\, BB \;\, P \;\, BB \;\, P \;\, BB = \{734136, 89656, 96640, 119368, 89232, 74048, 100680, 32112, 87080, 92064, 18336, 142008\}$$

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 differ from 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 criteria 1:

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

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. We continue by applying the criterion 2 on the P frames:

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

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 dropped first, P_2 second and P_1 will be the last one among P frames to drop. Since the value of P_3 is now the same as the values 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

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 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 largest). Assume now that we need to skip 4 frames. According to the assigned values, the skipping mechanism 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:

$$\begin{aligned} ESC_1: &\quad B_1 \to B_3 \to B_5 \to B_7 \\ ESC_2: &\quad B_2 \to B_4 \to B_6 \to B_8 \end{aligned}$$

We compare the total byte size in both chains, and we assign greatest values to the B frames in the chain with larger size:

$$\begin{aligned} size\ ESC_1: &\quad s(B_1)+s(B_3)+s(B_5)+s(B_7) = 229336 \\ size\ ESC_2: &\quad s(B_2)+s(B_4)+s(B_6)+s(B_8) = 402238 \end{aligned}$$

Since the total size of ESC_2 is larger than the size of ESC_1, we first decrease 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 is:

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 in the chain, that is 4. The final value assignment is:

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

So, the frame skipping according to the assigned values is performed as shown below:

													(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...

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.

VII. Timing Constraints

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. Therefore we do not include specific constraints, which would change with design decisions.

A. Start time constraints

The earliest time at which decoding a frame can begin is the earliest point in time at which all of the following start time conditions, *STC*, hold:

STC 1: *Frame header parsed and analyzed.*

STC 2: *For B and P frames: the decoding completion time of the forward/backward reference frame.*

STC 3: *Frame data available in input buffer.*

The cumulative input time CIT of a frame f_i, where i is the decoding number of f is calculated as:

$$CIT(f_i) = \sum_{n=1}^{i} \frac{FS(f_n)}{BR(f_n)}$$

with $FS(f_n)$ the frame size and $BR(f_n)$ the bitrate of frame f_n. $BR(f_n)$ and $FS(f_n)$ are available from the frame header

STC 4: *Free frame buffer available.*

This is always naturally true for reference frames: they require at least two buffers, one for the current frame and one for the previous reference frame it references to, see section IV. When a new reference frame is being decoded, at most one of them is needed for reference. As a consequence, for reference frames, STC4 becomes true one frame period earlier than it would for B frames.

The last two constraints are necessary for unblocked video stream processing.

B. Completion time constraints

The latest time at which decoding a frame has to be completed is the earliest point in time at which any of the following latest time conditions, *LTC,* holds:

LTC 1: *Required display time of the frame.*

If we have a TV set displaying a broadcast stream (DTV), the input frame rate is equal to the display frame rate: 50 - 60 Hz, depending on the region. Other input streams may have different frame rates, and other displays may have different display rates, i.e., the display rate is a multiple of the frame rate:

$$DR = \rho * FR$$

The frame period, T_{fr}, is equal to $1/FR$, while the display period, T_{dis}, is equal to $1/DR$. This means that ρ can be expressed as:

$$\rho = \frac{DR}{FR} = \frac{T_{fr}}{T_{dis}}$$

If the display rate is an integer multiple of the frame rate, i.e., ρ is an integer, the solution is simple, since the frame period and the display period will be harmonic. If this is not the case, things are more complicated. We will discuss both cases.

Case 1: *Display rate is an integer multiple of the frame rate:*

$$\rho \in Z^{+}$$

where Z^{+} is a set of positive integers (i.e., $\rho = 1, 2, 3, ...$).

Let f_i^j denote a frame with the decoding number i and the display number j. Note that, as outlined in section II-C, the decoding order will differ from the display order, i.e., $i \neq j$, if the stream contains B frames. For B frames $j = i - 1$, for I and P frames, the display number depends on the MPEG stream and has to be determined via look-ahead.

In this case, the required display time, RDT, of a frame with the decoding number i and the display number j is given by:

$$RDT(f_i^j) = IDL + (j-1)T_{fr}$$

where IDL stands for initial display latency, i.e., the display time of the first frame, as described in section IV-A. IDL includes "catching in" on the display period.

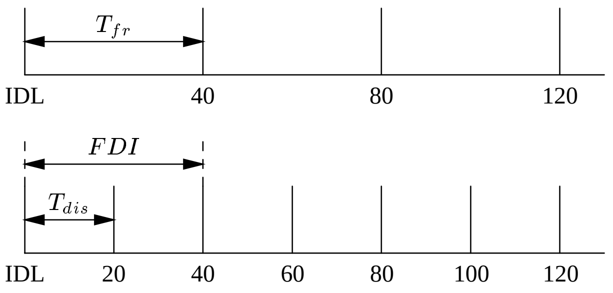
The length of the time interval in which a frame can be displayed is, in this case, the same for each frame, i.e., the length of the *frame display interval*, FDI, is equal to the frame period:

$$|FDI(f_i^j)| = T_{fr}$$

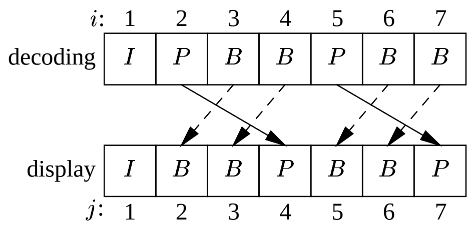
This implies that each frame will be re-displayed the same number of times, i.e., the repetition rate for the frames, R, is constant and it is equal to:

$$R(f_i^j) = \frac{|FDI(f_i^j)|}{T_{dis}} = \frac{T_{fr}}{T_{dis}} = \rho$$

Figure 12 depicts a simple example: assume an MPEG stream with a GOP structure $IBBPBBP$. If the frame rate is 25 fps, and the display rate is 50 fps, then we will have two invocations of the display task per frame. The frame period,



a) Frame rate and display rate, $DR = 2 * FR$



b) Decoding and display numbers of the frames

f	i	j	RDT	$\|FDI\|$	R
I	1	1	IDL + 0	40	2
B	3	2	IDL + 40	40	2
B	4	3	IDL + 80	40	2
P	2	4	IDL + 120	40	2
B	6	5	IDL + 160	40	2
B	7	6	IDL + 200	40	2
P	5	7	IDL + 240	40	2
...	...	...	...	...	...

c) Required display times, frame rates and intervals

Fig. 12. Case 1 - display rate is an integer multiple of the frame rate

T_{fr}, is equal to $1/25 = 40ms$, while the display period, T_{dis}, is equal to $1/50 = 20ms$, as shown in figure 12-a.

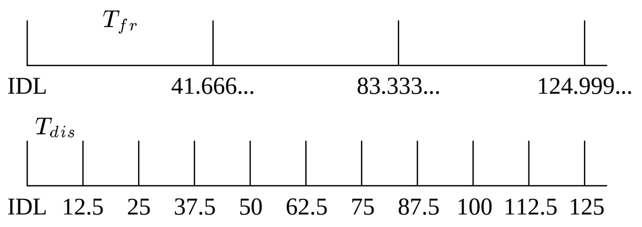
Decoding and display numbers are depicted in figure 12-b. For B frames, $j = i - 1$, e.g., the first B frame will have the decoding number 3 and the display number 2. For reference frames in this example, $j = i + 2$ (except for the first I frame in the stream which will have the same display and decoding number).

Finally, figure 12-c presents the corresponding frame intervals, repetition rates and required display times for the frames. Note different decoding and display numbers for the frames, e.g., the first P frame will have the decoding number 2, but its display time is 4, since we must display the two B frames first.

Case 2: *Display rate is not an integer multiple of the frame rate:*

$$\rho \notin Z^{+}$$

If the display rate is not an integer multiple of the frame



a) Frame rate and display rate, $DR = 3.333.. * FR$

f	i	j	RDT	$\lvert FDI \rvert$	R
I	1	1	IDL + 0	50	4
B	3	2	IDL + 50	37.5	3
B	4	3	IDL + 87.5	37.5	3
P	2	4	IDL + 125	50	4
B	6	5	IDL + 175	37.5	3
B	7	6	IDL + 212.5	37.5	3
P	5	7	IDL + 262.5	50	4
...	...	...	...	...	...

b) Approach 1: always pospone

f	i	j	RDT	$\lvert FDI \rvert$	R
I	1	1	IDL + 0	37.5	3
B	3	2	IDL + 37.5	50	4
B	4	3	IDL + 87.5	37.5	3
P	2	4	IDL + 125	37.5	3
B	6	5	IDL + 162.5	50	4
B	7	6	IDL + 212.5	37.5	3
P	5	7	IDL + 250	37.5	3
...	...	...	...	...	...

c) Approach 2: closest instance

Fig. 13. Case 2 - display rate is not an integer multiple of the frame rate

rate, than we can only find approximate solutions. Here is an example: assume that we have an input frame rate of 24 Hz (original film material), and a display rate of 80 Hz (computer display). The decoding period is proportional to the frame rate, i.e., $T_{dec} = 1/24 = 41.666...$ ms, whereas the display period is $T_{dis} = 1/80 = 12.5$ ms, as illustrated in figure 13-a.

Since the decoder task is not in phase with the display task, the required display times will not overlapp with starts of new frame periods, as in case 1. There are two ways to display frames:

Approach 1: Always postpone. The required display time for a frame is always *after* start of the corresponding frame period.

For example, required display time of the first B frame in the example from figure 13 (the one with $i = 3$ and $j = 2$) is equal to the start of the first display period that occurs *after* the start of B's frame period ($IDL + 41.666...$), which is $IDL + 50$. Similarly, RDT of the second B frame is the start of the first display period after $IDL + 83.333..$, which is $IDL + 87.5$ and so on, as shown in figure 13-b.

In this case, the required display time of the frames is calculated as:

$$RDT(f_i^j) = IDL + \lceil (j-1)\rho \rceil T_{dis}$$

Approach 2: Take the closest one. The required display time for a frame can be *before or after* start of the frame period, whichever is closest.

Let $\Delta_L(f_i^j)$ and $\Delta_R(f_i^j)$ denote the time distance from the start of f's frame period to the closest left respective right start of display period, i.e.,:

$$\begin{aligned}\Delta_L(f_i^j) &= (j-1)T_{fr} - \lfloor (j-1)\rho \rfloor T_{dis} \\ \Delta_R(f_i^j) &= \lceil (j-1)\rho \rceil T_{dis} - (j-1)T_{fr}\end{aligned}$$

The required display time for this approach is given by:

$$RDT(f_i^j) = IDL + \begin{cases} \lfloor (j-1)\rho \rfloor T_{dis}, \text{ if } \Delta_L(f_i^j) < \Delta_R(f_i^j) \\ \lceil (j-1)\rho \rceil T_{dis}, \text{ otherwise} \end{cases}$$

For example, for the first B frame, $\Delta_L(f_3^2) = 41.666 - 37.5 = 4.166$ and $\Delta_R(f_3^2) = 50 - 41.666 = 8.333$. Since Δ_L is less than Δ_R, the required display time is equal to $IDL + 37.5$ and not $IDL + 50$, as we would have in approach 1. Required display times for the other frames is shown in figure 13-c.

The repetition rate for the frames (both in approach 1 and approach 2) will not be constant for each frame, since the frame display intervals will have different length. For example, if we use approach 1, $FDI(f_1^1)$ in the example above will have length 50, while $FDI(f_3^2)$ will have length $87.5 - 50 = 37.5$.

The length of the frame display interval for the both appraches in this case is equal to the required display time of the frame that is to be displayed next, i.e., the one with the display number $j+1$ (and some decoding number k):

$$|FDI(f_i^j)| = RDT(f_k^{j+1}) - RDT(f_i^j)$$

The repetition rates are calculated as:

$$R(f_i^j) = \frac{|FDI(f_i^j)|}{T_{dis}}$$

Approach 1 is a little more relaxed in terms of precise latencies, and thus deadlines. Apparently, the choice between approach 1 and 2 does not really matter with respect to relative frame jitter. In both cases, we get a cycle of three frame intervals: 50, 37.5, 37.5. However, the relative frame jitter is important for perception. In high quality video where the jitter is not accepted, this problem has been solved by using interpolation, i.e., making new frames. This feature is called *natural motion* [18].

LTC 2: *Imminent overflow of input buffer*.

By a judicious choice of input buffer size, as outlined in section IV, LTC2 will always be met. Should the completion constraint be missed, though, data loss at the input buffer will occur, with the risk of having to recapture the stream, which will take at least the complete GOP or until the next sequence header.

VIII. Conclusion

In this paper, we presented a study of realistic MPEG-2 video streams and showed a number of misconceptions for software decoding, in particular about relation of frame structures and sizes. Furthermore, we identified constraints imposed by frame buffer handling and discussed their implications on timing constraints.

Using the analysis, we determined realistic flexible timing constraints for MPEG decoding that call for novel scheduling algorithms, as standard ones that assume average values and limited variations, will fail to provide for good video quality.

Our current work includes extending the study to the sub frame level, e.g., relationship between framesize and execution time, motion vectors, and sub frame decoding. Furthermore, we are formulating a quality based frame selection algorithm to be used in a real-time scheduling framework.

Acknowledgements

The authors wish to thank the reviewers for their fruitful comments which helped to improve the quality of the paper. Further thanks go to Clemens Wüst and Martijn J. Rutten from Philips Research Laboratories, Eindhoven, for their careful reviewing and useful comments on this paper.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] J. Watkinson, *The MPEG handbook*. ISBN 0 240 51656 7, Focal Press, 2001.

[9] L. Teixera and M. Martins, “Video compression: The mpeg standards,” in *Proceedings of the 1st European Conference on Multimedia Applications Services and Techniques (ECMAST 1996)*, Louvian-la-Neuve, Belgium, May 1996.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] G. de Haan, “IC for motion compensated deinterlacing, noise reduction and picture rate conversion,” *IEEE Transactions on Consumer Electronics*, August 1999.

Damir Isovic is lecturer and graduate student at System Design Lab, Departement of Computer Science, Malardalen University, Sweden . He received a Master of Science in Computer Engineering (1998), a Diploma of Higher Education in Natural Science Mathematics and Astronomy (1999) and a Technical Licentiate in Real-time Systems (2001), all from Malardalen University. Damir’s research interests include real-time systems and scheduling theory, with a specific emphasis on combining flexibility and reliability in construction of schedules, real-time components and real-time multimedia.

Gerhard Fohler is professor and leader of the predictably flexible real-time systems group at the Depatement of Computer Science, Malardalen University. He received his Ph.D. from Vienna University of Technology in 1994 for research towards flexibility for offline scheduling in the MARS system. He then worked at the University of Massachusetts at Amherst as postdoctoral researcher within the SPRING project. During 1996-97, he was a researcher at Humboldt University Berlin, investigating issues of adaptive reliability and real-time. Gerhard Fohler is currently chairman of the Technical Committee on Real-Time Systems of EUROMICRO.

Liesbeth F. Steffens graduated in physics from Utrecht University in 1972, took a post-graduate computer-science course at Grenoble University, and joined Philips Research in 1973, where she spent most of her career. Her main focus is on real-time, quality-of-service, and resource-management issues for streaming and control in consumer-electronics devices and systems. She contributed to the design of a distributed real-time operating system, a video-on-demand system, and a QoS-based resource-management system for digital video