The Implementation of PET

Bernd Lamparter Malik Kalfane [*]

TR-95-047

August 1995

Abstract

This report describes the implementation of PET (Priority Encoding Transmission) and its integration into VIC [6]. PET [1] is a new Forward Error Correction (FEC) scheme with several priority levels. It is useful for applications dealing with real-time transport streams like video and audio in a lossy environment. Those streams consist of several data parts with different importance. PET allows to protect those parts with appropriate redundancy and thus guarantees, that the more important parts arrive before the less important ones.
Keywords: Video Transmission, Forward Error Correction, MPEG, Parallelizing.

[*]both International Computer Science Institute, Berkeley, CA 94704, USA

Contents

1 Introduction

Using advanced coding techniques, PET improves the quality of transmission over packet networks that have unpredictable losses. PET encodes and stripes the information from long messages into many shorter packets. PET encodes the data using a unique multi-level Forward Error Correction scheme that provides graceful acquisition and degradation of information.

PET makes possible something that has not been practical due to bursty losses: high quality video transmission over lossy packet networks (such as Internet) using MPEG. These bursty losses have forced the data communication industry to broadcast video using motion JPEG, which requires several times the transmission rate of MPEG. PET, by allowing the use of more efficient techniques such as MPEG, will greatly reduce the overall transmission rate requirements despite the fact that it introduces a minimum increase in the MPEG transmission rate.

In chapter 2 we describe the striping of the message into packets and chapter 3 defines the corresponding format of the PET packets. Many modern operating system support threads to make usage of a multiple processor or to enhance the software architecture. PET is build thread safe, i. e. applications can use threads without the risk of screwing up internal data structures of PET. Threads are also used internally by PET to make usage of multiple processors (see chapter 4). Chapters 5 and 6 are describing the incorporaton of MPEG and PET into VIC. Some enhancements planned for the future are described in chapters 7 and 8.

The algorithm for the actual generation of the redundant information is beyond the scope of this report and may be found in [2].

2 Striping of the Message into packets

Given a message consisting of message parts with corresponding priorities, the PET algorithm must map this onto packets in such a way that the PET guarantees are satisfied: for instance, if the user requests that a given message part of size 10KB be sent with priority 80%, then the message must be encoded in such a way that from any 80% of the packets sent, that part of the message will be recoverable.

PET first appends the priority table to the beginning of the message, since it must be sent with the message, and creates a new priority table that contains an extra entry corresponding to the priority table and its priority. Then the different message parts are sorted according to their priority, and message parts with identical priorities are concatenated, so that roundoff errors in satisfying the priorities will be minimized. This yields yet a new priority table, with shorter length, since some message parts will be concatenated, and with entries sorted from lowest priority (i.e. highest redundancy) to highest priority. The decoder will be able to reconstitute the original message, as the original priority table is sent with the message, and from this table, all the modifications to the priority table can be reconstructed.

The final priority table is the input to the striping algorithm, which also takes as input the length of the packets, and the length of each packet segment. A packet segment is the basic element used by the erasure code to encode the message. For instance, in the case of the polynomial scheme, it is just two bytes, which is the space required to store an element of the finite field $GF[2^{16}+1]$. In the case of the Cauchy scheme, the segment size is four bytes times the dimension of the finite field as an algebra over $GF[2]$. The larger the segment size, the larger the roundoff errors will be in trying to satisfy the priorities for the message, as the packet can be broken up into fewer segments.

Since a whole number of segments must be assigned to each priority level, a fraction of the last segment for each level might be wasted because of roundoff error. Additional information could be sent for this priority level without using any more segments, by adding enough information to fill the wasted fraction of the last segment. In the striping algorithm, we compute the number of segments needed for the lowest priority level first. We then determine how much extra information could be sent without using any more segments, and we send that amount of information from the next priority level together with the current message part. This means that some of the information from the second priority level will be sent at a lower priority level than what was required by the user, which does not hurt the PET guarantee, as it means that this information will be recovered from fewer packets than the user had asked for. We then decrease the size of the second message part by the amount we sent at the lower priority level, and we keep on doing this for each priority level. Hence, for all but the last priority level, we make full use of each segment of the packets, by sending less important information at a lower priority level than required. This helps us lower the roundoff errors that occur in trying to satisfy the priority table.

Given k message parts, each of size L_i bytes and of priority P_i, where P_i is expressed as a floating point number between 0 and 1, and $P_i < P_{i+1}$, and a packet consisting of S segments of length l bytes each, the striping algorithm is as follows:

1. Compute the theoretical encoding size, which is the size needed to encode the message if there were no roundoff errors. The encoding size is given by $E = \sum_{i=1}^{k} \frac{L_i}{P_i}$. This gives the minimum number of packets needed to encode the message as $N = \lceil E/(S \times l) \rceil$.

2. Starting at $N = \lceil E/(S \times l) \rceil$, and incrementing N by 1 until the priority table can be satisfied, do the following steps:

   (a) For i ranging from 1 to k, do the following steps:

      i. Compute the number of message packets for this priority level $m_i = \lfloor P_i \times N \rfloor$, so that the associated message parts will indeed be recoverable from any P_i fraction of the packets.

      ii. Compute the number of segments needed for this level: $s_i = \lceil L_i/m_i \rceil$.

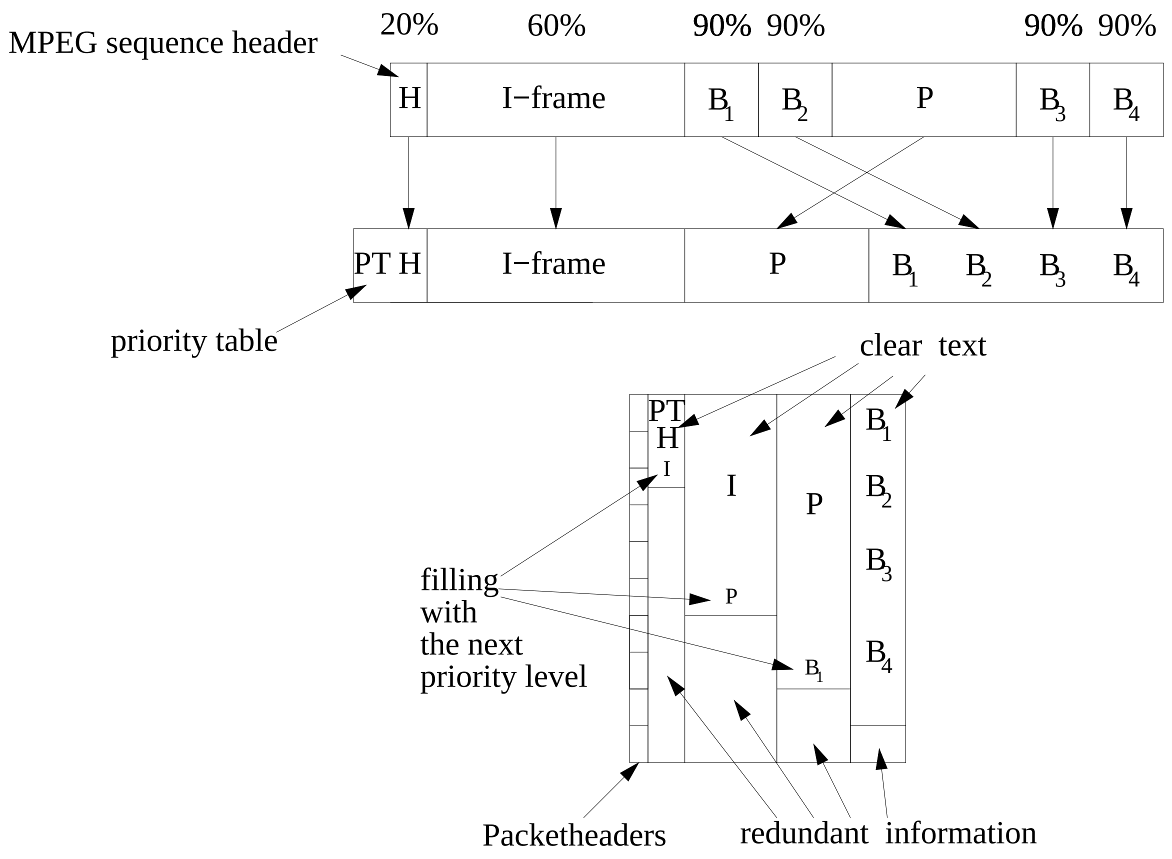


Figure 1: Sorting and striping of messages

iii. The total amount of information that can be sent with this number of segment is $s_i \times n_i$, so send $(s_i \times n_i) - L_i$ info from the next level, and decrease L_{i+1} by this amount. If $i = k$, i.e. if we are at the last level, then no information can be sent from the next level, so some of the segment will be wasted.

(b) Compute the total number of segments used: $T = \sum_{i=1}^{k} s_i$. If $T > S$, then all the priority levels cannot be satisfied with this number of packets.

3 Packetformat

The format of each packet follows the striping processes described in the previous chapter. Each PET packet (see fig. 2) consists of a header and the priority levels. The first priority level is special, because it has the priority table at its beginning. To find out the values of the other priority levels when decoding a message the priority table has to be extracted first.

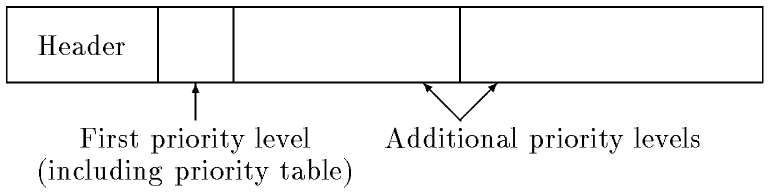


Figure 2: PET packet format

Each header contains the necessary information to decode the first priority level: **n_packets** (total number of packets), **pp_tab** (priority of the priority table) and **priol** (priority table length). As soon as enough packets have arrived, the first priority table can be decoded. The first part of this decoded message block is the priority table.

Even if there are not enough packets to decode the first priority level completely, it is sometimes still possible to decode the priority table: The encoding scheme guarantees that the message itself is in the first packets as clear text, whereas the redundant information is in the last packets. Usually the priority table contains too few bytes to fill the first priority level completely. If the first few packets have arrived, we find the priority table as cleartext in those packets. This gives the application at least the type of each of its priority levels.

The header of each PET packet contains the following information (see fig. 3):

version The PET version used.

ID An identifier for the message. The sender of the PET packets is responsible to provide a unique identifier for each message send. Because there are only 256 different IDs available, the protocol between sender and receiver must insure that the receiver has never an old message with the same ID in his memory. In rare cases of very high losses it may happen, that the receiver gets confused by two message with the same ID. Therefore PET performs a few consistency checks on the received packets. If one of the fields **n_packets**, **pp_tab**, or **priol** or the packet length doesn't match, PET refuses to accept the packet. The user should then abort the current message and hand the packet again to PET.

seq_no A sequence number within the current message.

n_packets The total number of packets in this message. We need this and the next two values to extract the priority table.

pp_tab The priority of the priority table.

priol The length of the priority table.

version	ID	seq_no	n_packets	pp_tab	priol

Figure 3: Format of the header of a PET packet

4 Multithreading support and parallelizing

Many modern UNIX variants have support for multiple threads. This threads run in parallel within one UNIX process. This gives advantages for many applications even if there is only one processor in the workstations. Those applications may delegate a job (for example PET encoding) to a separate thread and in the meanwhile process other events (e.i. updating of the user interface). Those applications may even call functions of the PET library while another call is in progress. PET has to protect himself against the potential danger to the integrity of his data.

Additionnally PET itself may take advantage of multiple processors in a machine. It may compute priority levels or packets in parallel.

4.1 Protection against reentry

If several threads manipulate the same set of data, inconsistencies may occur. Therefore we need to protect each set of data in a way, that only one thread is working on each.

There are two basic mechanism for synchronization in multithreaded environments: mutual exclusion (mutex) and semaphores. mutex-operations are simpler to use and faster, but they protect only certain pieces of code against concurrent execution. PET uses several mutex for protection of the following code parts:

- No protection is used on the encoding side, since there are only two functions (PET-numberOfPackets and PETencode) and they are called one after the other. There is no oncurrency possible. Applications may create a thread to execute PETencode, but there is no second call possible on the same data.
- There is one mutex for each decoder. It protects the creation and lookup of messages.
- Each message is protected by an own mutex. This mutex is the most important one, because applications may start threads for decoding as soon as one priority level is available. Then the application may add more packets to the message as they arrive. Therefore the procedure **PETprocessPacket** and the critical parts of **PETretrieveMessage** are protected by one mutex. In this way it is possible to add more packets to a message without interfering with the decoding process.

Whereas a mutex is used to protect a certain data set, semaphores are more general. It is a general mechanism for synchronization between several threads. In PET we use semaphores to signal the end of a thread to several other threads.

4.2 Parallelizing of the encoding and decoding

PET computes a matrix consisting of packets and the segments of the packets, where the later packets consist of the redundant parts. Hence there are basically three possibilities to parallelize encoding and decoding of PET messages:

- Compute different priorities in parallel.
- Compute a bunch of packets in parallel.
- Parallelize the computation of the segments of each priority. That's an extension of the first approach.

The best speedup is reached by the second approach, because the different tasks are equal in computational effort and the linear part of the algorithm is small. The drawback is, that some values will be calculated in each thread newly. A second problem occurs when the according priority level contains only a few segments. The overhead in creating threads may be too large.

We implemented the first and second approach for encoding, and the first one for decoding only.

4.2.1 Compute the priority levels in parallel

The different priorities are already computed in a separate function call, so we simply had to run this calls in parallel. We start a new thread for each call and then wait for the end of them. The problem with this approach is, that the CPU time for each of this threads is rather different. So some threads are ending earlier and hence only a part of the CPUs (eventually only one) can be kept busy. Let's consider an example for this behaviour:

- 4 processors
- 6 priority levels with 1, 2, 3, 4, 5 and 6 secs CPU-usage

On a one processor machine this computation would take 21 secs. But what is happening on our 4 processor machine? The first 1.5 secs all six tasks are running on four CPUs, then it runs another 1.2 secs with 5 threads, after another second one CPU gets out of usage. The last second only one CPU is busy. That gives as a speedup of only $21/6.7 \approx 3$ (see fig. 4). A better scheduling of the tasks would help, as long as the number of task is as

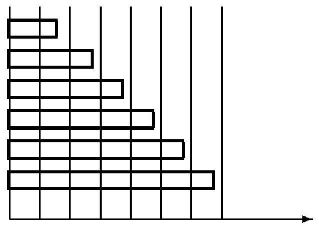


Figure 4: Usage of four CPUs

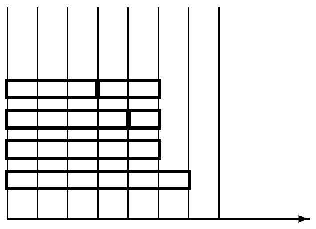


Figure 5: Better usage of four CPUs

large as the number of CPUs (see fig. 5). In the example we have a speedup of $21/6 = 3.5$. That is optimal, when we assume, that this tasks are undividable.

The implementation of the parallelization of the different priorities is straight forward. But on the decoding side, things are more complicated. The application may start several decoding threads where it wants to have different message parts. That means, that PET has to take care of already running threads, which are decoding one or more priority levels.

PET uses the following algorithm:

```
mutex_lock
    for each priority level
        if the application wishes a message part of this level
        and no thread is working on it, then
            start a new thread to decode this level
mutex_unlock
for each priority level
    if the application wishes a message part of this level
        if we started the thread
            wait for it
            for each other thread waiting for this priority level
                free the semaphore
        else
            wait for the semaphore of the thread
mutex_lock
combine the message parts into the memory provided by the application
mutex_unlock
```

Whenever an incarnation of PETretrieveMessage() finds a priority level with an attached thread, it increments a specific counter. Later it waits for a specific semaphore. The incarnation of PETretrieveMessage(), which started the according thread, frees the semaphore according to the counter and with that all waiting incarnations of PETretrieveMessage() are restarted.

4.2.2 Compute the packets of one priority level in parallel

To encode a priority level we have to produce two types of packets: In a first step we copy the message into the clear text packets and in the second we compute the redundant packets. Both steps can take advantage of multiple processors, if there are enough packets to produce and the priority level has enough packets.

This approach is also possible for decoding, but it is much more complicated to implement.

5 An Application for PET: MPEG in vic

The first effort to apply PET to MPEG was an off line demonstration [5]. The PET version used there was too slow to be applied to a real-time MPEG decoder. With the newer version which uses cauchy matrizes instead of polynomials to produce the redundant information it was possible to run PET in real-time.

VIC is a popular application for interactive video developed by Steve McCanne. VIC stands for video interactive conferencing [6]. It consists of several encoding schemes (NV, h.261, JPEG, CuSeeMe) and runs on workstations of all major vendors.

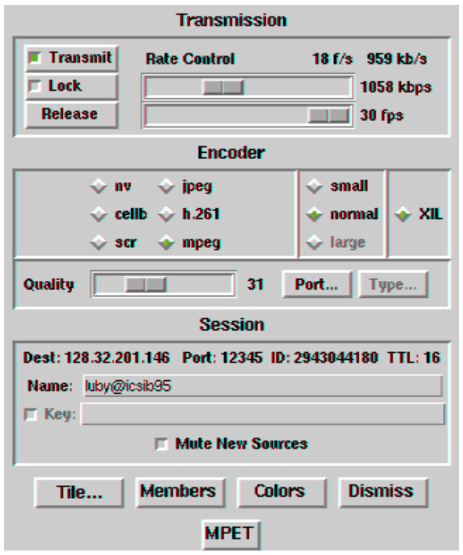
5.1 MPEG decoding

As first step we built MPEG into VIC. The decoding process is done in software with a modificated version of the MPEG player developed by Stefan Eckhart and the MPEG Software Simulation Group at the University of Munich, Germany.

Th complete player was translated from C to C++ and all global variables became class members of the MPEG decoder class. The old programm was only able to play one MPEG stream at a time, whereas in VIC several incoming MPEG stream are possible. The second change was the buffering mechanism. The old version reads a chunk from file and stores it in a buffer. When this chunk has been consumed, a new chunk was read. That assumes an independent comsumer process, which can read more data whenever necessary. In VIC the decoding process is driven by incoming data not by the consumer. VIC stores incoming packets in a buffer and as soon as a frame is complete, it starts the MPEG decoder to decompress this frame.

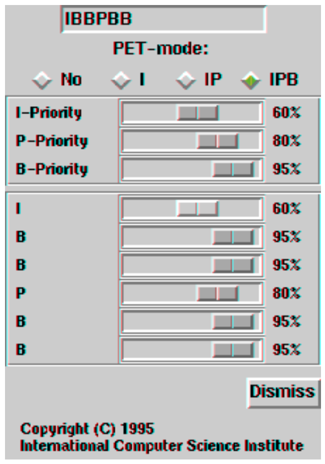
5.2 MPEG encoding

For the encoding we use the sunvideo board. With the GUI of VIC (see fig. 6) the user can specify the desired frame and bitrate. An additional window (see fig. 7) allows the user to set the pattern of the MPEG frames and various PET parameters.



18 f/s 959 kb/s
1058 kbps
30 fps
Quality 31
Dest: 128.32.201.146 Port: 12345 ID: 2943044180 TTL: 16
Name: luby@icsib95

Figure 6: vic GUI



IBBPBB

I-Priority	60%
P-Priority	80%
B-Priority	95%
I	60%
B	95%
B	95%
P	80%
B	95%
B	95%

Copyright (C) 1995
International Computer Science Institute

Figure 7: The MPET window

5.3 RTP encapsulation of plain MPEG streams

5.3.1 Header fields

The MPEG packets are RTP encapsulated [7], but currently uses none of the MPEG specific header fields. It is planned to support RTP-encapsulation as defined in [3]. Currently the MPEG specific header is four bytes long, non is set, and all ignored.

5.3.2 Payload

According to [3] the frame should be fragmented in packets at the beginning of slices, so that the MPEG decoder can immediately resume decoding after a packet loss. This results in packets of different size. The framer in VIC fills all packets despite the last one, but the reassembler don't expect the packets to have equal length.

5.3.3 Loss Recovery and handling of late packets

Losses are handled differently according to the type of the affected frame. Hitten B-frames are discarded whereas I- and P-frames are handed to the decoder as they are. As soon as the decoder finds out, that something is wrong (e.i., he didn't find the end marking of a macro block), he will skip to the next slice start. This means, that B- and P-frames may reference to wrong data. But skipping all wrong data means to skip the whole GOP if only one packet of the I-frame is lost. In a lossy transmission this could result in a complete stop.

If packets are arriving too late (see Chap. 6), that might be a sign, that the decoder is too slow to keep up with the incoming data stream. Eventually that will result in packet losses due to an overflow of the buffer for incoming packets. It is hence important to reduce the computational burden on the CPU. Frames with at least one "nonlate" packet are considered to be in time and decoded. "late" B-frames are discarded, I- and P-frames are decoded anyway. This scheme gives I- and P-frame a higher chance to survive but may result in more hitten B-frames.

5.4 RTP encapsulation of PET protected MPEG: MPET

The optional RTP header for PET is defined to have four bytes:

type	subtype

The type field in the RTP header is set to PET (33). The type of the encoding in the optional header is used as defined in [7]. The subtype is additional information on the exact enocding used. In MPEG encoding we use currently three different schemes:

PET_I Encodes the I-frame only.

PET_IP Encodes the I- and the P-frames, but each as a separate message.

PET_IPB Encodes the whole GOP in one PET message. That is the usually used version despite its large delay.

5.4.1 Timing: When to play a frame

Figure 8 shows the timing for a MPEG stream with the frame pattern IBBPBB. The frames are grabbed and compressed to the appropriate time (or read from file) and send to the framer. The framer stores the frames until the next GOP header is received. Then it starts the PET encoding process. If the encoding is short enough or done in a separate thread, we will not miss to grab the next frame. After encoding of the PET packets, they are send out spaced, i. e. they are not send all at once, because that would lead to a highly bursty traffic. Instead the sending times are spread over the time for the next GOP. The framer sets the timestamp in all packets to the time when the last frame was captured. [1]

The receiver may start the decoding when he receives the first packet of the next GOP. He can then calculate the difference of the timestamps for the actual and the next GOP and hence find the times between the frames. It then PET-decodes the GOP and start a Tcl-timer to initiate the MPEG decoding and displaying of the frames.

The MPEG decoding of the single frames is initiated by the Tcl-timer routine. First we check, if we are already behind the schedule. In this case and if we have a B-frame to decode, we just discard the frame and restart the Tcl-timer. If we have a P-frame and we are only

[1]Unfortunatly the XIL-library refuses to give us the real capture time. So we ask the operating system for this time.

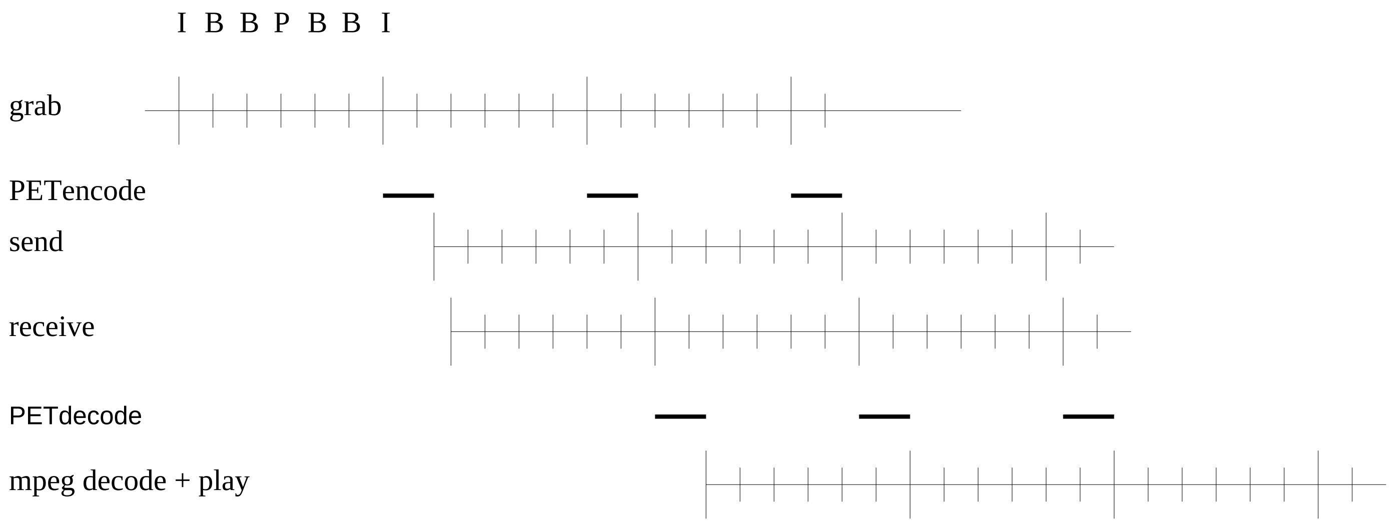


Figure 8: Timing of MPEG frames

one frame behind we decode and display it. If we have a I-frame, we decode and display it in any case. With this scheme we avoid overloading of the CPU which would result in a loss of packets in the incoming buffer. After decoding and displaying of a frame the time for the next frame is calculated and the Tcl-timer restarted. No new timer is started after the last frame of a GOP.

In PET mode all packets are equally important, the “late” flag is ignored. Still we have to make sure, that the buffer is emptied frequently. The simplest way would be, to give the corresponding Tcl-routine a high priority. Unfortunatly, the Tcl event mechanism has no priority scheme. Hence we make sure, that the timing mechansim leaves at least a few milli seconds till the next frame is due. The Tcl scheduler calls the packet reader in this gap and empties the buffer.

6 When are packets too late?

RTP has a field for a timestamp set by the sender. Hence it is possible to compute an average delay and mark packets which arrived a certain value later as “late”. On the other hand, the main concern is not the delay in the network, late packets occur whenever the CPU is not able to process packets fast enough. Packets will then be buffered and read too late. A “real” late calculation would only be possible, if the operating system would tag packets as soon as they come in from the network interface.

If we are simply interested, whether we are behind the schedule or not, we can have a look on the buffer length. After reading a packet from the queue we look if there are more packets available. In this case the packet is forwarded with “late”-flag set and the next packet is read immediately thereafter.

The decoder can now decide, whether to drop the current frame or to decode it and risk the loss of a packet and hence hurt a frame.

7 Future Work on PET

We need to do more experiments to find out, how many parallel threads should be started to compute a single priority level. There are several parameters to determine this number: Available processors, size of the clear text and redundancy data area (e.i. the number of packets in each category times the number of segments), computing power of the CPUs and the overhead of creating a thread.

The creation of a thread is rather cheap ($\approx 50\mu$secs according to the handbook), but there is a cheaper possibility: It is a client/server modell with multiple servers for the encoding. A dispatcher is then responsible to hand the encoding task to the next free server thread and we need a mechanism to inform the clients when the task is done. All that can be done with semaphores which are cheaper to create and to deal with compared to the creation of a new thread.

8 Future Work on Applications for PET

Currently we use the prioritizing only for MPEG GOP coding. Another scheme prioritizes the DCT matrix coefficients according to their level of importance. This approach is also useful for JPEG images or movies. With the current scheme, the video becomes jerky in the presence of losses. With this approach, the video rate is smooth, but instead the quality of single frames varies according to the losses.

An advantage of the DCT prioritizing is the reduced delayed compared to the GOP prioritizing. But to split the DCT matrix means to decode the huffman encoding, split the matrix, and huffman encode the two or more parts. This is not only an computational effort but will also result in a coding overhead because of multiple "End of Block" tags of the DCT-matrix.

A third, natural scheme could be applied if the encoding is layered in subbands, like i. e. the 3-d subband coding [8]. The layer can easily be mapped to the PET priority levels.

References

[1] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. In *Proceedings of 35th Annual Symposium on Foundations of Computer Sciences, Santa Fe, New Mexico*, pages pp. 604–612, Nov. 1994.

[2] Johannes Blömer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. An xor-based erasure-resilient coding scheme. Technical report, International Computer Science Institute, Berkeley, California, 1995.

[3] Don Hoffman, Gerard Fernando, and Vivek Goyal. Rtp payload format for mpeg1/mpeg2. Technical report, IETF, 1995. ftp://ds.internic.net/internet-drafts/draft-ietf-avt-mpeg-00.txt.

[4] Bernd Lamparter, Andres Albanese, Malik Kalfane, and Michael Luby. PET - Priority Encoding Transmission: A New, Robust and Efficient Video Broadcast Technology. In *Proceedings of ACM Multimedia '95*. ACM, November 1995.

[5] Christian Leicher. Hierarchical Encoding of MPEG Sequences Using Priority Encoding. Master's thesis, International Computer Science Institute and Technische Universität München, Nov. 1994. ftp://ftp.icsi.berkeley.edu/pub/tenet/PET/tr-94-058.ps.Z.

[6] Steven McCanne and Van Jacobson. *vic:* a flexible framework for packet video. In *Proceedings of ACM Multimedia '95*. ACM, November 1995.

[7] Henning Schulzrinne, Steven Casner, Ron Frederick, and Van Jacobson. Rtp: A transport protocol for real-time applications. Technical report, IETF, 1995. ftp://ds.internic.net/internet-drafts/draft-ietf-avt-rtp-06.9.ps.

[8] D. Taubman and A. Zakhor. Multi-rate 3-d subband coding of video. In *IEEE Transactions on Image Processing*, April 1993.