Robust Transmission of MPEG Video Streams over Lossy Packet-Switching Networks by using PET*

Andres Albanese[1] and Giancarlo Fortino[1,2]

[1]International Computer Science Institute, Berkeley, CA, USA
[2]University of Calabria, Rende (CS), Italy

TR-99-014

June 1999

Abstract. Network heterogeneity is a major issue and multimedia applications have to deal with interconnected networks consisting of many sub-networks of non-uniformly distributed resources. Real-time traffic caused by video sources is bursty by nature, resulting in buffer overflow at the switch and unavoidable packet losses. Therefore the information is desirable be compressed and prioritized in a way that the application gracefully degrades during adverse network conditions. Priority Encoding Transmission (PET) is an approach to the transmission of prioritized information over lossy packet-switched networks. The basic idea is that the source assigns different priorities to different segments of data, and then PET encodes the data using multilevel redundancy and disperses the encoding into the packets to be transmitted. The property of PET is that the destination is able to recover the data in priority order based on the number of packets received per message. This report summarizes the results to date obtained from the PET project and gives direction of on-going and further work. The paper describes the fundamentals of the theory on which PET is based, the integration of PET with MPEG-1, some experimental results, and an application tool RTP-based, VIC-MPET, which allows encoding and playing robust MPEG video streams over the Internet MBone.

* Research supported by the National Science Foundation operating grant NCR-9416101.

Table of Contents

Table of Figures

Index of Tables

1. Introduction

Multimedia computing enables new applications such as video on-demand, multimedia email, documents and spreadsheets, desktop video conferencing, distributed virtual reality, and distributed interactive simulation. Being a powerful tool it will have a dramatic impact on social, educational and business communications:

"*The transition to distributed multimedia applications will be more significant than the transition to graphical user interfaces because it will have a greater impact on business productivity and our personal lives.*" (Prof. Larry Rowe, University of California at Berkeley, 1995)

By introducing optical fiber to transmit data a tremendous increase in network bandwidth has been achieved, allowing widespread use of multimedia applications. Wide area networks of global dimensions offer ubiquitous access to users not only in the research community but also increasingly to businesses. Therefore network heterogeneity has become a major issue and applications have to cope with interconnected networks consisting of many sub-networks. Computational power, bandwidth, storage and congestion control policies may unevenly vary in different network environment resulting in unavoidable congestion, delays and losses.

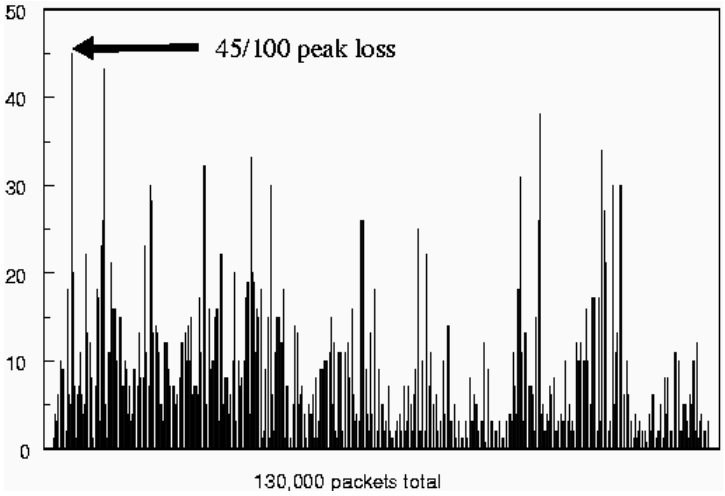

Figure 1.1: Packet losses in a multicast video session.

Figure 1.1 shows the packet loss per 100 measured at one receiving host during a network video multicast session (generated by the multimedia tool *nv*[1]) over the Multicast Backbone

[1] ftp://ftp.parc.xerox.com/net-research

testbed (MBone[2]) from Palo Alto to Berkeley. Note that the peak loss is 10 times greater than the average loss of 4.5%.

The transmission of images, moving or still, requires an enormous amount of bandwidth [Crowcroft et al., 99]. Efficient compression algorithms have been developed but still network traffic will be easily dominated by the video. Contrary to computer data traffic, a set of mechanisms is required in order to provide good quality of service (QOS) guarantees. Strong delay constraints challenge the design and engineering of communication systems. Circuit switching technology imposes constant bit-rate (CBR) utilization of the assigned channels. Thus video encoders are forced to generate a CBR stream, paying the price of varying picture quality and unsatisfying bandwidth exploitation.

Packet-switched networks such as the Internet and wide area ATM networks provide a basis for variable bit-rate (VBR) video transmission and constant picture quality. In ATM networks VBR coding schemes may be implemented without sacrificing bandwidth utilization efficiency, since bandwidth can be allocated on demand. However, VBR coding and transport introduces other potential obstacles.

Video traffic is bursty by nature. Thus when many sources are transmitting at their peak rate the available network bandwidth may be exceeded, causing buffer overflow at the switch and packet loss. Thus, packet loss is more likely to occur in bursts, rather than be evenly distributed. Although some loss of less important visual information may be acceptable, the introduction of compression algorithms using source coding techniques and motion compensation may lead to severe degradation in terms of picture quality when losses occur. Encoded bitstreams using variable length codes (VLC) are very sensitive to losses, since the loss of a single bit forces the decoder to discard information until the following synchronization sequence. In order to guarantee a minimum quality of service, priority mechanisms are desirable. The Moving Pictures Expert Group (MPEG) has developed a widely used standard for video compression. Although MPEG-1 does not provide scalability, it may also be viewed as hierarchically structured. Reference frames (intra frames) carry the complete set of parameters needed for frame reconstruction at the decoder, whereas other frames (inter frames) contain only information about changes from these reference frames. However, some efforts have been made adding scalability in terms of resolution to the MPEG-1 bitstream. A two layered scheme over ATM has been proposed, prioritizing packets in two levels: base layer and enhancement layer. The base layer contains only low frequency DCT (Discrete Cosine Transform) coefficients and can be decoded

[2] http://www.mbone.com

independently, whereas the enhancement layer carries high resolution data that is only usable when added back to the base layer. By separating the bitstream the bit rate per frame is increased by approximately 20%. The network has to make sure that the base layer is transmitted over a more reliable channel.

Several other approaches introduce network support in order to cope with packet loss. One idea is based on retransmission of discarded packets. Techniques such as Automatic Repeat Request (ARQ) have significant drawbacks, since they introduce additional roundtrip latency and in a multicast scenario cause enormous complexity (e.g., scalability issues).

Alternatively, codes based on Forward Error Correction (FEC) traditionally focus both on detection/correction of errors and recovery of lost information. In a packet-based network, error detection/correction can be dealt with on a packet by packet basis. Thus, the primary use of FEC by the application would be to recover lost packets. Traditional FEC permits encoding only on a single priority level.

Priority Encoding Transmission (PET) [Albanese et al, 96] is an approach regarding the prioritized transmission of scalable messages over lossy packet-switched networks. It allows a user to specify priorities for different segments of data. Based on this hierarchical structure the sender uses the system to encode the data into packets for transmission. The idea is that the information is distributed over all the packets sent per message. However, each packet contains relatively more information about the higher priority segments of the data. Hence the receiver is able to recover the information in priority order, based on the number of packets received per message.

This report addresses the robust transmission of MPEG video streams over MBone by using PET techniques. Its main goal is to summarize and describe the research and the results obtained from the PET project. Section 2 is aimed at providing basic information about the MPEG-1 standard. In section 3 a PET system using erasure coding techniques based both on properties of (1) polynomials over a finite field and (2) Cauchy matrices is described. Section 4 merges PET and MPEG. A way of packetizing MPEG-1 bitstreams is proposed. The basic idea is that inter-frames (B) are less redundantly encoded than reference frames (I and P). A PET-MPEG-1 simulation based on the first PET scheme is detailed by commenting observed results with different MPEG movie clips. Section 5 introduces the VIC-MPET multimedia tool and the PET-API. Conclusions and directions of on-going work are finally presented.

2. MPEG

MPEG[3] is a compression standard for audio, video, and data established by the International Telecommunications Union and International Standards Organization. The MPEG-1 standard [ISO11172], established in 1992, is designed to produce reasonable quality images and sound at low bit rates (e.g., 352x240 (288) images at 24-30 fps with VHS quality at 1.5 Mbps). The MPEG-2 standard, established in 1994, is designed to produce higher quality images at higher bit rates (e.g., 720x485 studio quality CCIR-601 images at up to 15 Mbps).

Its development therefore addressed the following features:

Random Access, Fast Forward/Reverse Searches, Reverse Playback, Audio-Visual Synchronization, Editability, and Format Flexibility

The basic idea behind MPEG video compression is to remove spatial redundancy within a video frame, as in JPEG (the standard for still image compression spatial redundancy), and temporal redundancy between video frames. Motion compensation is used to exploit temporal redundancy. The images in a video stream usually do not change much within small time intervals. The idea of motion-compensation is to encode a video frame based on other video frames temporally close to it.

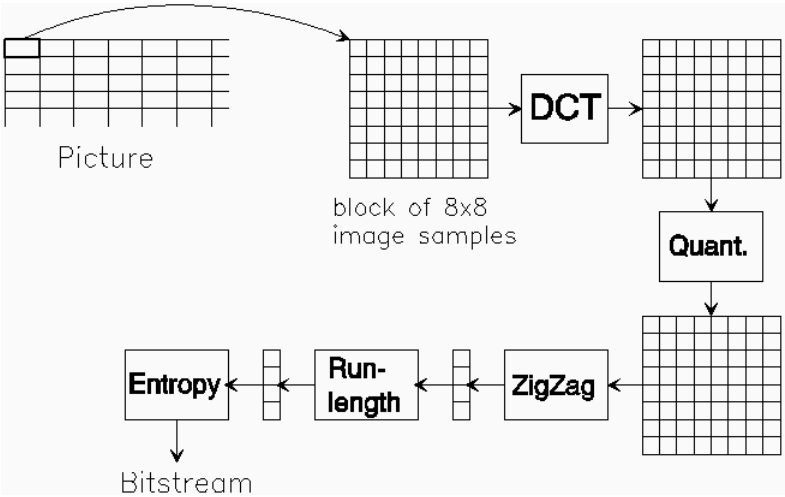

Figure 2.1: Block Transfer Encoding, the basis of JPEG and MPEG encoding.

Image compression usually is achieved by applying several techniques such as sub-sampling of chrominance components, quantization, frequency transformation by the cosine transform

[3] http://www.mpeg.org/MPEG/

(DCT - Discrete Cosine Transform) and variable length coding (VLC) (see Figure 2.1). MPEG additionally introduces motion compensation (MC) to exploit temporal redundancy, predictive coding and picture interpolation.

2.1 MPEG-1 Video Compression

A video stream is a sequence of video frames. Each frame is a still image. A video player displays one frame after another, usually at a rate close to 30 frames per second (23.976, 24, 25, 29.97, 30). Frames are digitized in a standard RGB format, 24 bits per pixel (8 bits each for Red, Green, and Blue). MPEG-1 is designed to produce bit rates of 1.5Mbps or less, and is intended to be used with images of size 352x288 at 24-30 frames per second.

The MPEG-1 algorithm operates on images represented in YUV color space (Y Cr Cb). If an image is stored in RGB format, it must first be converted to YUV format. In YUV format, images are also represented in 24 bits per pixel (8 bits for the luminance information (Y) and 8 bits each for the two-chrominance information (U and V)). The YUV format is sub-sampled. All luminance information is retained. However, chrominance information is sub-sampled 2:1 in both the horizontal and vertical directions. Thus, there are 2 bits each per pixel of U and V information. This sub-sampling does not drastically affect quality because the eye is more sensitive to luminance than chrominance information.

Sub-sampling is a lossy step. The 24 bit RGB information is reduced to 12 bit YUV information, which automatically gives 2:1 compression. Technically speaking, MPEG-1 is 4:2:0 YCrCb.

2.1.1 Coding Layers

As illustrated in Figure 2.2 an MPEG bitstream is built up of four major components. Pictures are the equivalent of a single movie frame and represent the basic unit of display. There are three different kinds of pictures, which are discussed later in more detail. A frame is divided into slices.

The slices are the basic processing unit and provide fault tolerant access within a single picture. Since they are independent, they may be coded in groups. A slice may be a single line of the image or consists of any number of macroblocks. Each of them contains a header and six component blocks: 4 luminance blocks and just 2 chrominance blocks (Figure 2.3).

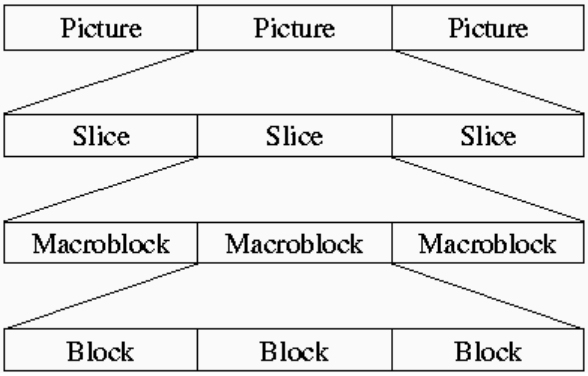

Figure 2.2: MPEG coding layers.

Due to the characteristics of sensitiveness of the human eye, color information is sub-sampled. That is, a macroblock of 16x16 pixels represents 4 blocks of luminance Y, but only one block of chrominance blue Cb and one of chrominance red Cr. Squares of 8x8 pixels each are called blocks, the smallest coding entity in the MPEG algorithm. Macroblocks are the units for motion-compensated compression. Blocks are used for DCT compression.

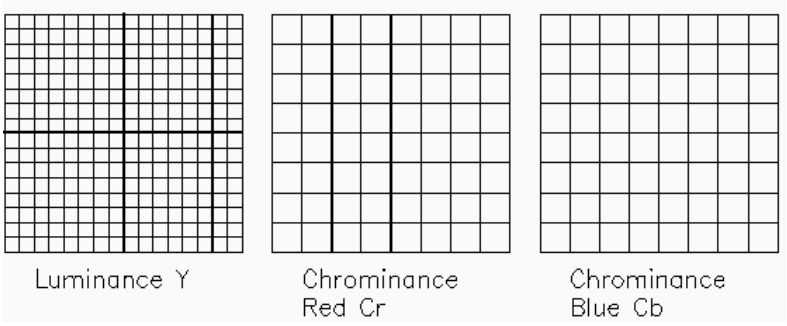

Figure 2.3: A macroblock in MPEG-1.

2.1.2 Coding Techniques

MPEG uses three different types of pictures:

- Intra-frames (I), coded as a still image;
- Predicted frames (P), predicted from the most recently decoded I-or P-frame;
- Bidirectional-frames (B) interpolated from the closest two I-or P-frames.

An *I-frame* is encoded independently from any other image (past or future frames), applying techniques similar to JPEG compression each 8x8 block is first transformed from spatial domain into the frequency domain using Discrete Cosine Transform (DCT), which separates the signal into independent frequency band. Most frequency information is in the upper left corner of the

resulting 8x8 block. After this, the data is quantized. The corresponding coefficients then are quantized in frequency order that is low frequency components are encoded more accurately than high frequency ones. Quantization is the only lossy part of the whole compression process other than sub-sampling. Additional compression is achieved by variable length coding (RLE - Run Length Encoding) of the resulting data in a zig-zag ordering. Each block is encoded independently with the following exception: the coefficient in the upper left corner of the block, called the DC coefficient, is encoded relative to the DC coefficient of the previous block (DCPM coding).

P-frames generally refer to the most recent reference frame (past frame), which is either an I- or a P-frame. Each macroblock in a P-frame can be encoded either as an I-macroblock or as a P-macroblock. An I-macroblock is encoded just like a macroblock in an I-frame. A P-macroblock is encoded as a 16x16 area of the past reference frame, plus an error term. They use motion compensation on a macroblock basis. Encoded are motion vectors and error terms. The vector specifies the relative location of the macroblock within the reference frame, which matches the one to be coded best. A motion vector (0, 0) means that the 16x16 area is in the same position as the macroblock being encoded. The difference is expressed by an error term or in case of total compliance is skipped. In the latter case the reference macroblock is simply duplicated. Motion vectors may include half-pixel values, in which case pixels are averaged. The error term is encoded using the DCT, quantization, and run-length encoding. The range for motion vectors may be limited, since searching for the closest pattern is very time consuming. The search for good motion vectors (the one that gives small error term and good compression) is the heart of any MPEG-1 video encoder and it is the primary reason why encoders are slow.

A *B-frame* may be interpolated from past and future reference frames. So B-macroblocks may either use backward motion compensation (MC), forward MC or both, in which case the 16x16 area is averaged (Figure 2.4). Bi-directional frames therefore provide the highest amount of compression, but may not be suitable for all applications, since they require out of sequence transmission, which leads to latency.

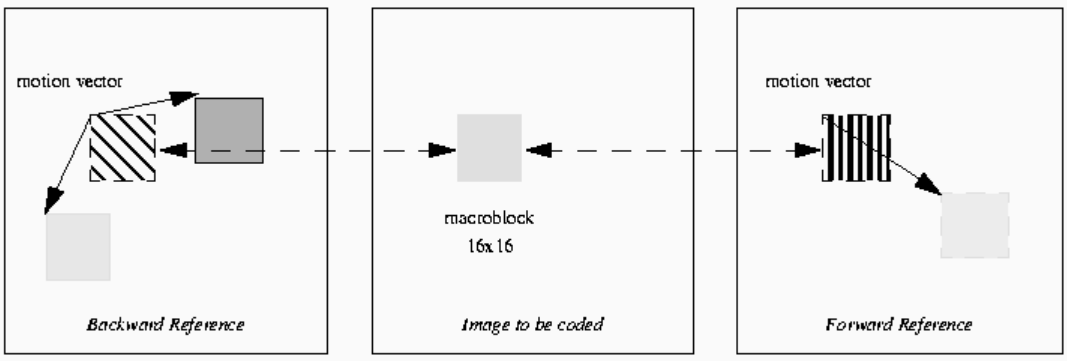

Figure 2.4. Motion Compensation (MC) in P- and B-frames.

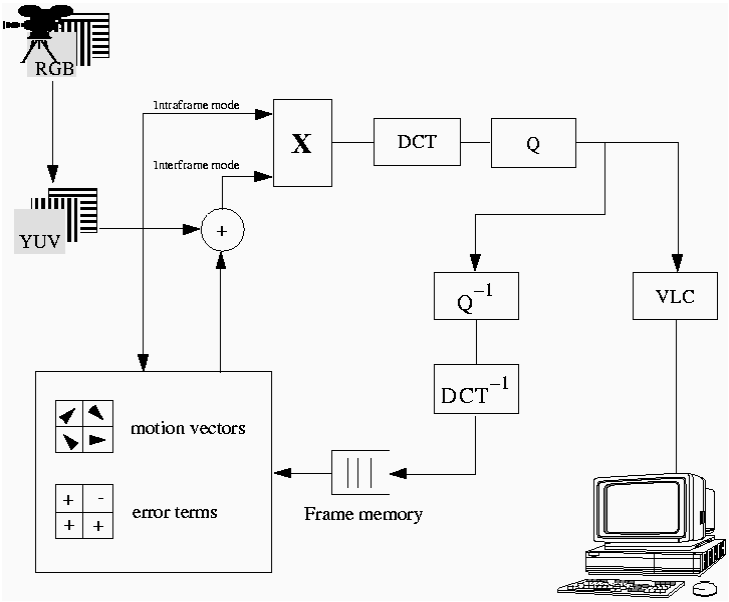

Figure 2.5. Main MPEG coding loop.

Figure 2.5 shows the complete coding loop for the MPEG algorithm. Note that the feedback loop is only needed for P-and B-frames.

2.1.3 Bitstream Hierarchy

An MPEG video is represented by a layered and ordered bitstream. The top layer is called sequence. Sequence delimiters encapsulate a movie. It always starts with a sequence header which contains essential information such as picture size, picture rate and bit rate. Eventually it ends with a 32 bit sequence end code.

The header is followed by any number of Group of Pictures (GOP). A GOP (fig. 2.6) provides random access, since it is the smallest coding unit that under certain circumstances can be independently decoded.

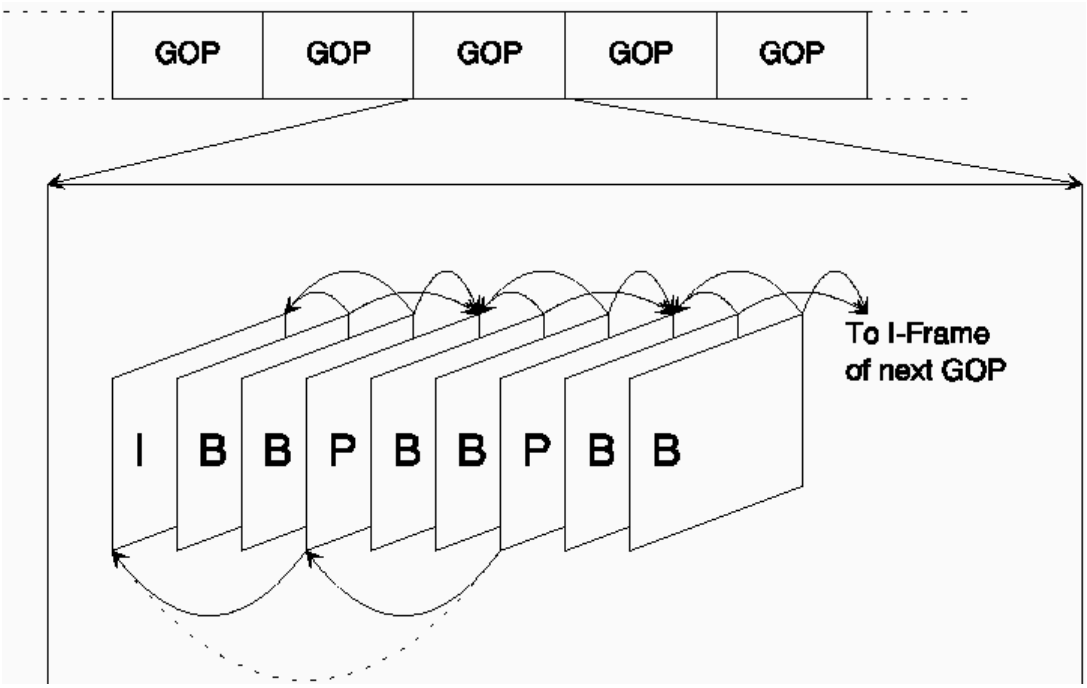

Figure 2.6. MPEG-1 video stream.

Due to the variety of picture types, the design of a GOP is constrained by the following properties [ISO11172]:

- Property 1. A group of pictures, in bitstream order must start with an I-frame and may be followed by any number of I-, P-or B-frames in any order.
- Property 2. It must begin, in display order, with an I-or B-frame and must end with an I-or P-frame. The smallest GOP might consist of only a single I-picture, whereas the upper bound is unlimited.
- Property 3. It always begins with a GOP-header and either ends at the next GOP-header or at the end of sequence.

Note that in case a GOP starts with a B-frame in display order, it cannot be decoded independently, since it requires a reference frame of the previous GOP. In general, a GOP visualizes the dependencies between the several types of frames within a GOP or even across their borders. As mentioned before, there are two different picture orderings: display order and

bitstream order. The latter has to be different, since for interpolation the decoder first has to know about the two reference frames, before he might evaluate the macroblocks within a B-frame slice.

Note that in bitstream order frames can only refer to past reference frames. In case the first block of B-frames is interpolated, it depends on the P-frame of the previous GOP. The MPEG standard provides a closed GOP flag. It denotes whether the GOP is open or closed. In the latter case a GOP can be decoded without any references to previous groups. With respect to the transmission of MPEG sequences over packet-switched networks, the closed GOP flag could slightly increase the fault tolerance of the bitstream.

Figure 2.7 shows the difference between bitstream order and display order. Frames usually carry temporal references that denote the display order. In general bi-directional frames change places with reference frames. The very first GOP varies from the subsequent, since two references are needed in order to decode the first block of B-frames. However, the I-Frame and the P-Frame at the beginning is a kind of initialization and therefore usually is not displayed at all.

Another point worth mentioning concerns the size of the frames. There is no such thing as a fixed size for all three frame types. The I-frame size can vary in the different GOPs of a video sequence depending on the scene content. The sizes of P- and B-frames can vary even within a GOP. In addition the MPEG-1 standard allows the pattern of I-, P- and B-frames to vary dynamically between different GOPs although this is usually not done.

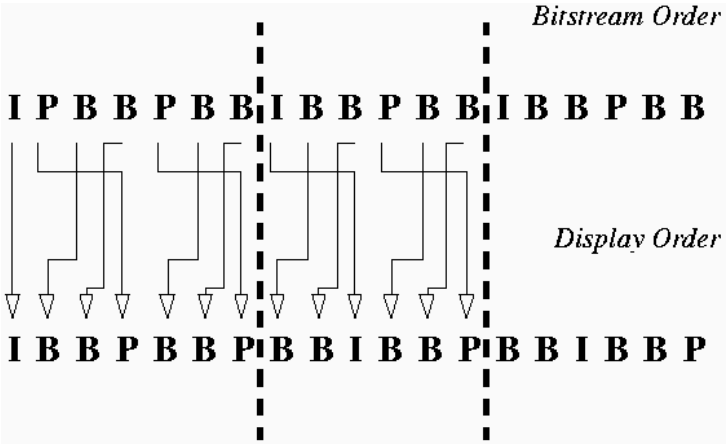

Figure 2.7. Bitstream order vs. Display order.

3. PET (Priority Encoding Transmission)

Priority Encoding Transmission (PET[4]) [Albanese et al., 96; Albanese et al., 96; Lamparter et al., 95] is an approach introducing a method for sending messages over a lossy packet-switched network according to a user specified prioritization scheme. It focuses on fault tolerant transmission, especially suitable in case of unpredictable packet losses. Its design is independent from any specific scalable application. The basic idea is that the source assigns different priorities to different segments of data resulting in a multilevel redundancy distribution. The information is distributed over all the packets sent per message. However, each packet contains relatively more information about high priority data. Hence the destination is able to recover the information in priority order based on the amount of packets received per message.

In this chapter, two PET systems are described. The first uses erasure coding techniques based on polynomials over a finite field. The latter exploits erasure coding techniques based on Cauchy matrices. In the first paragraph the notion of erasure codes is briefly introduced. The second paragraph describes basic properties of polynomials over finite fields, whereas the third one introduces the Cauchy matrices. Finally, a possible implementation of a PET system is highlighted.

3.1 Erasure Codes

Let us assume a message M consists of b words of length w each. Consequently the message length m adds up to m = wb. Message M is redundantly encoded into code E(m) with length e = nw, where $n \geq b$. E is called an erasure code if the original b words can be reconstructed from any b words of the encoded message E(m) (together with the indices of the b words of encoding) [Luby et al., 97].

Figure 3.1 displays a message M and the corresponding code E. The b words can be recovered from any subset of b words of the total encoding. Two examples depicted by the continuous/dashed line are shown in Figure 3.1.

A possible implementation of erasure codes consists in viewing the b words as the coefficients of a polynomial of degree b-1 over a finite field. This method is described in the following sections.

[4] http://www.icsi.berkeley.edu/~fortino/PET/index.html

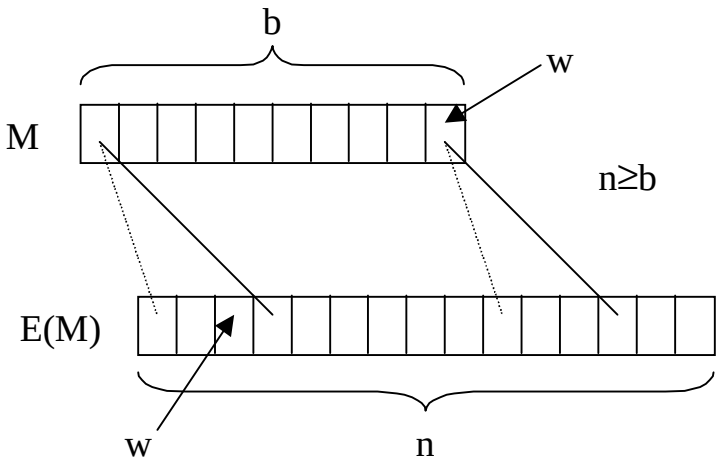

Figure 3.1. Erasure codes.

3.2 Polynomials over a Finite Field

3.2.1 Fundamentals on the Galois Field

There are two different types of finite fields or Galois fields (GF) [Albanese et al., 96]: Prime fields GF[p] and extension fields GF[p^m] where "p" denotes a prime number and "m≥2" a positive integer. Respectively they consist of q = p or q = p^m elements, which in the latter case can be considered as polynomials of degree m-1 with coefficients $a_i \in$ GF[p]:

$$a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1}$$

In Galois field GF[p] operations are calculated modulo p, whereas in Galois field GF[p^m] an irreducible polynomial p(x) is needed, so calculation is done modulo p(x). The arithmetic in extension fields is more complicated than in prime fields. Therefore an erasure code implemented in a prime field was considered to be more suitable for the proposed purposes. One way to realize erasure codes consists in viewing the b words of message M as the coefficients of a polynomial of degree b-1 over a suitable chosen field. Code E then is built up by n polynomial values evaluated at different field elements. Hence, from any subset of b polynomial values (and the corresponding field elements at which they are evaluated) the original message can be determined by interpolation.

As an example it is assumed that 4 integer values (1,8,5,3) should be transmitted. Since all 4 values are less than 9, they can be considered as elements of Galois field GF[9]. Now the 4-tuple (1,8,5,3) denotes the coefficients of the polynomial p(x) mod 9. It is obvious that from any 4 values p(i) evaluated at different field elements i the original polynomial p(x) can be reconstructed by interpolation. The redundancy is set by transmitting more than 4 pairs (i,p(i)). Consequently in Galois field GF[9] a maximum of 9 different pairs might be created. All calculations are done modulo 9.

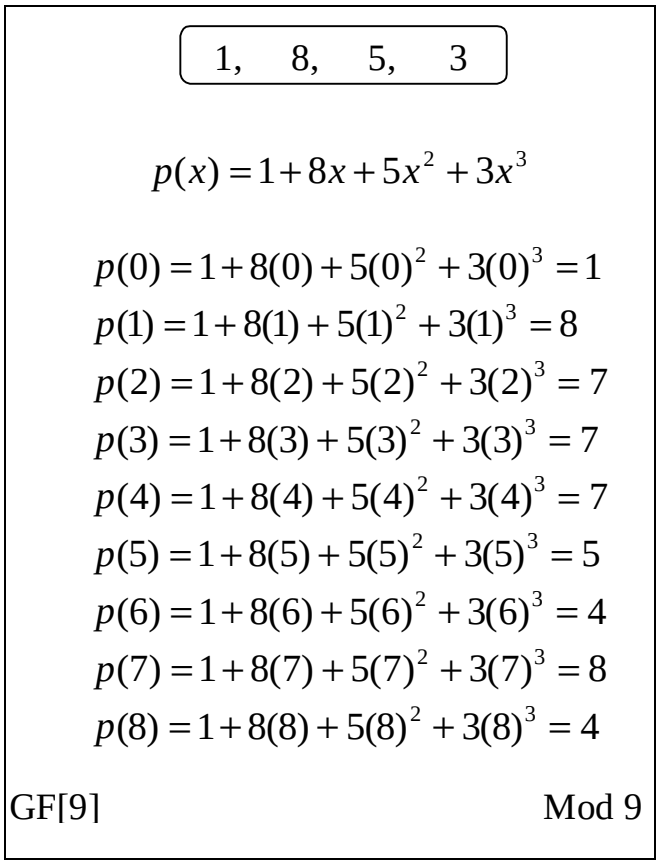

Figure 3.2. Polynomials over Galois field GF[9]

3.2.2 A possible Galois Field for a PET System

Assuming the system uses words of length w = 16 bit or 2 bytes each, they might be considered as coefficients of a polynomial over the extension field GF[2^{16}]. However, from a computational point of view it is preferable to use a prime field. The Galois field GF[2^{16}+1] is a prime field covering the range that can be represented by 16 bit words. Multiplications within this field can be reduced to a constant number of integer arithmetic operations, shifts and comparisons.

However, the value is not representable by 16 bits, but obviously may be one of the polynomial values evaluated at different field elements. Therefore in order to avoid overflow a PET system has to provide an additional feature.

In case the list of transmitted polynomials contains the element, an offset ε has to be defined which makes sure that the range representable by two bytes is not exceeded. The offset per packet is determined by browsing the transmission data for a value that is not a member of the data within the same packet. Then all values smaller than the offset $x < \varepsilon$ are mapped onto themselves $x = x$ and all values $x > \varepsilon$ are mapped onto $x = x-1$. Consequently within each packet, 16 bits have to be reserved for the offset.

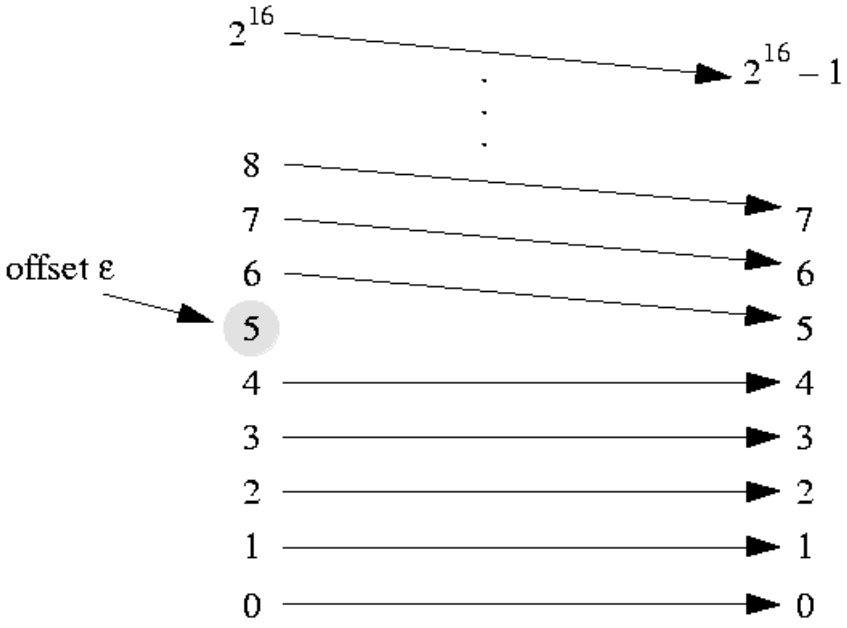

Figure 3.3: Representation of Galois field GF[2^{16}+1]

3.3 Cauchy Matrices

Cauchy matrices [Bloemer et al., 95] can be used to implement linear, systematic codes, i.e., the encoding is a linear function of the message and the unencoded message is part of the encoding. MDS codes based on Cauchy matrices are a variant of Reed-Solomon codes [*IETF draft FEC*].

Packets contain b words consisting of w = 32 bit. For a code over GF[2], a message of m packets is then considered to be an (mbx32)-matrix M over GF[2].

Since typically the encoding size is only a constant multiple (0..5) of the message size the last property helps implementing an efficient encoding procedure for these codes. On the encoding side running time is achieved that decreases with the amount of unencoded message received. Since often the encoding is moderately larger than the message itself, this is one of the reasons the decoding procedure runs in real-time for bit rates up to a few Mbps.

However, the main gain compared to the method based on the evaluation and interpolation of polynomials is achieved by using a well-known representation of elements in finite fields of the form $GF[2^L]$ by (LxL)-matrices over GF[2]. This allows to replace arithmetic operations on field elements by XOR's of computer words and in practice XOR's of computer words are much more efficient than multiplications in finite fields.

3.4 Basic Design Considerations

3.4.1 Data Partitioning

The first step in constructing a PET system is partitioning the transmission data [Leicher, 94]. These portions are the basic units, called messages which are encoded one at a time. A message then is split into user specified priority segments. A segment is divided into blocks. Each block represents a polynomial of degree (block length) -1. Within each segment all blocks should have the same length in order to meet the given priority. Then each block is encoded separately by using erasure codes. The n polynomial values of the erasure code E(m) are dispersed into n packets. Each packet therefore contains only one polynomial value represented by a single word. The field element varies from packet to packet, but within a single packet all polynomials are evaluated at the same field element.

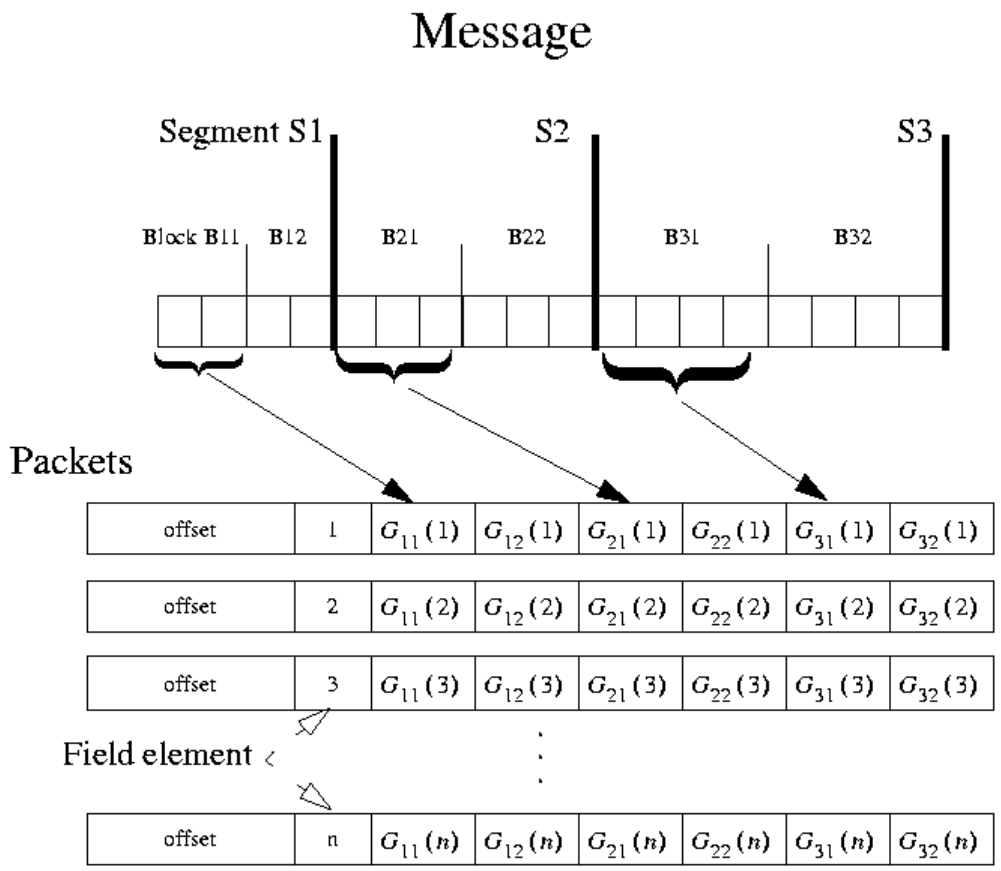

Figure 3.4. Message striping process.

Figure 3.4 shows an example message partitioned into three priority segments and striped into packets of equal length. Note that the maximum number of packets which can be sent is $2^{16}+1$, the size of the Galois field GF[$2^{16}+1$].

The smallest entity is a word w of two bytes. The priority of a segment is expressed by the length of its blocks. Hence the smaller the blocks, the higher the priority. Obviously in our sample message segment S1 has the highest priority, followed by S2 and S3. Consequently blocks B11-B12 consist of two, B21-B22 of three and B31-B32 of four words. $G_{sb}(i)$ denotes the value of the corresponding polynomial of degree equal to (block size) -1, evaluated at the field element i (indices s and b identify the segment and block respectively). Within each packet all polynomials are evaluated at the same field element. Hence B11-B12 can be reconstructed from any two packets, B21-B22 from any three and B31-B32 from any four packets. By sending n=6 packets B21 now would be encoded with 50% redundancy, i.e., it can be recovered from any 50% of the transmitted packets. Each packet additionally carries at the beginning the offset discussed above and the field element i.

3.4.2 The Priority Table

One of PET's basic design principles is that the priority function is specified by the user or the application. Based on this distribution the message is divided into segments. A simple and flexible realization consists in specifying priorities in fraction of packets needed to recover the segment. Regarding the previously discussed example (fig. 3.5), a possible user defined table may look as follows:

Segment size (word)	Fraction of packets needed to recover
S1=4	0.333
S2=6	0.500
S3=8	0.667

Table 3.1. Priority table information.

The number of transmitted packets (n) is set to 6. Consequently even though segment S1 encompasses only 22.22% of the total message it covers a third of the encoding. Since any pair of packets is already sufficient to recover S1, an overall overhead of 200% for S1 is sent. The corresponding values for the other segments are listed below in priority order:

Segment size	Fraction of message size	Transmitted redundancy
S1=4	22.22%	200%
S2=6	33.33%	100%
S3=8	44.44%	50%

Table 3.2. Information distribution.

The total encoding length adds up to two times the length of the original message, not included the overhead for the offset, field element and the priority table.

4 Prioritized Encoding of MPEG Sequences

4.1 Motivation

Priority Encoding Transmission (PET) [Albanese et al., 96] is designed to be an independent interface between a packet-switched network and any scalable application. The latter has to provide a priority scheme according to which PET is able to assign redundancy. In this chapter a scenario which should demonstrate the feasibility of the design considerations is introduced and simulation results with MPEG sequences are described. The focus of this work is on the recovery of information sent over a lossy medium. It has to be pointed out once again that PET is not only limited to this proposed application but is also aimed at coping with any scalable data.

From an information theoretic point of view MPEG video compression represents a source coding scheme. Even though PET does not provide error correction, it can be considered as a kind of channel encoding. Compared to systems based on Forward Error Correction (FEC) codes, PET uses less redundancy overall, since the redundancy is unevenly distributed. Concerning the network, a single channel reservation is proposed. In case the data is already layered, it should be multiplexed onto a single channel (fig. 4.1).

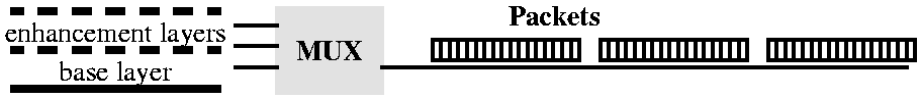

Figure 4.1. Multiplexing of layered data onto a single channel.

Due to the rapid development of multimedia applications, there is an increasing need for video compression techniques. So far, most applications use Motion JPEG, which is considered to be more fault tolerant. Only by introducing P and B frames, MPEG provides higher compression rates than JPEG, but at the same time makes the bitstream even more vulnerable to losses:

Figure 4.2 displays a MPEG-1 bitstream affected by lost packets. Note how the image is mixed up from several frames. Macroblocks are not correctly replaced or use the error term in combination with an affected reference frame. Usually affected macroblocks might only vary slightly from the original, but their blocky character distorts the image.

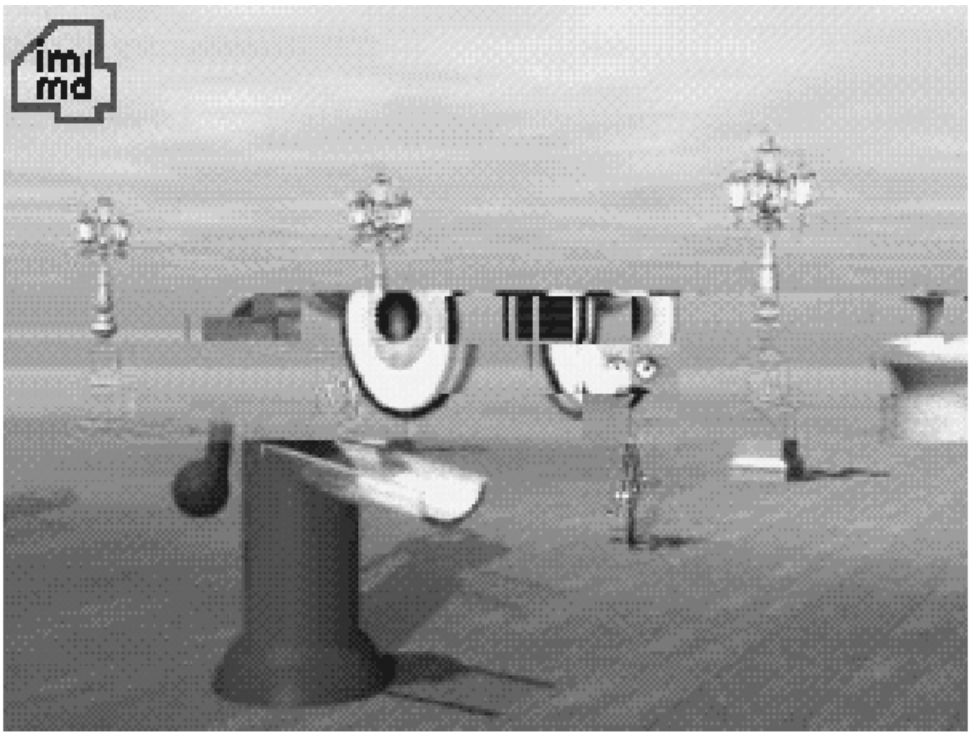

Figure 4.2. Losses affecting MPEG.

The MPEG video compression standard is considered not to be suited for transmission over packet networks, since it was designed for storage of video. However, protected by a PET system, requirements for transmission over a packet-switched network could be met by increasing the bitstreams fault tolerance. Video traffic is bursty by nature and therefore affects the stability of packet networks. In order to cope with losses there is an increasing demand for scalable video. Scalable extensions have been added to the succeeding MPEG-2 standard, but do not promise to be very effective. MPEG-1 has become very popular and many software as well as hardware coders are emerging. It does not provide scalability on a picture quality basis, but the dependencies between different types of frames provide a priority scheme for PET. Since the goal was a feasibility proof of PET as an independent interface, coding techniques, such as customizing MPEG encoders do not need to be changed. Even though the granularity in this case might be limited to a frame basis, MPEG-1 provides a good basis for a preliminary PET simulation.

4.2 Segmentation of MPEG Sequences

A MPEG (see §2) video sequence is composed of sequence headers and groups of pictures (GOP). Header information usually has to be especially protected. It might be transmitted only once at the very beginning and end of a sequence. It not only identifies a bitstream as a MPEG movie, but also contains the following essential parameters set by the encoder:

- Horizontal/Vertical Picture Size
- Pel Aspect Ratio
- Picture Rate
- Bit Rate
- Constrained Parameter Flag.

As shown in Figure 2.6 losses within an I-frame at least affect a whole GOP and in most cases as well the first block of B-frames of the succeeding GOP. Therefore it is obvious that within a GOP an I-frame has to have highest priority. This segment should also encompass GOP header information, since without the corresponding I-frame it is of no use either. B-frames require an additional reference frame in order to be correctly decoded. This reference usually is a P-frame which itself is dependent on the previous I-frame. Consequently P-frames should have higher priority than B-frames.

A possible way to encode MPEG is to prioritize over a whole GOP and map it onto a PET message. At the very beginning the message might also contain sequence header data. This solution has a certain drawback, since several frames have to be buffered first and therefore latency is added to the system. Hence in this version it might not be suited for interactive video conferencing applications, but several other scenarios such as broadcasting of conferences and Video on-Demand (VoD) systems should be taken into consideration. Nevertheless in case of a small number of frames in a GOP, it might also be applicable for interactive usage.

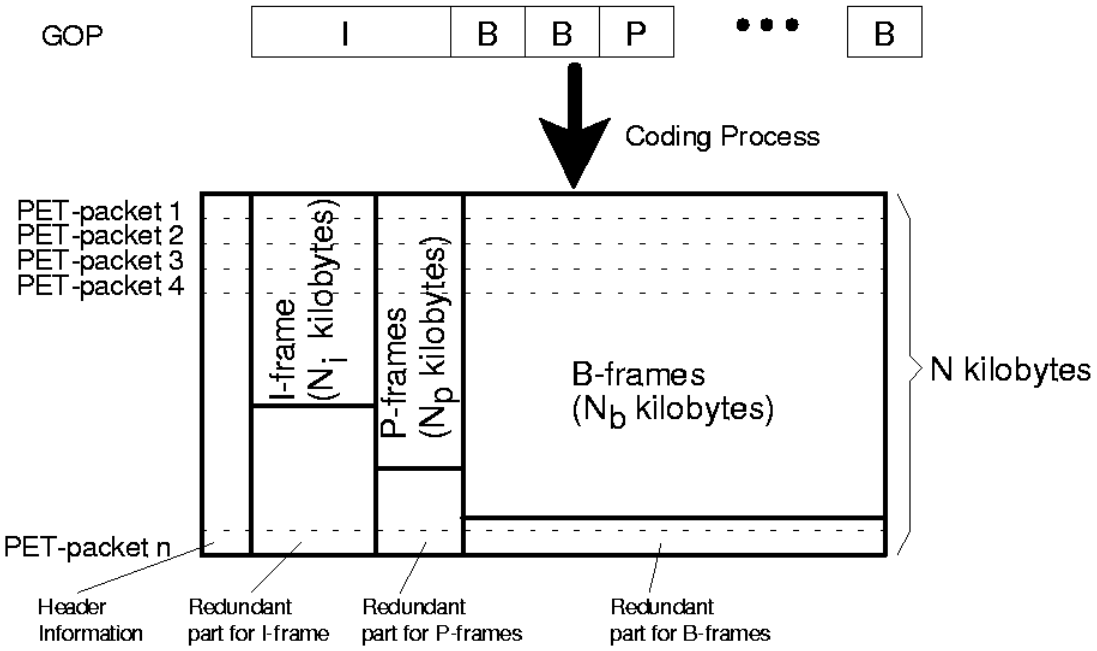

Figure 4.3. Illustration of the coding process that is effective in PET.

4.2.1 Redundancy distribution

A typical MPEG-GOP frame pattern is considered:

I B B P B B

Regarding a picture size of 320x240 pixels, Table 4.1 shows representative frame sizes:

Frame Type	Size in Bytes	Occurrences in GOP	Total Size	Fraction Needed	Encoding
I-Frame	12000	1	12000	60%	20000
P-Frame	4800	1	4800	80%	6000
B-Frame	2850	4	11400	95%	12000
Total		6	28200		38000

Table 4.1. Typical frame sizes.

Priorities are expressed by fraction of packets needed to recover the original information. According to the above considerations, the I-frame might be encoded in a way that it can be recovered from any 60% of the total number of packets sent, the P-frame from any 80% and the

two blocks of B-frames from any 95% of the packets. The very first message includes a fourth priority segment, which contains important header data. It might be encoded highly redundantly, so it can be reconstructed from any 10% of the packets.

Figure 4.4 displays this multilevel encoding example. Note how the total overhead is unevenly distributed over the data. Although the original message is 74.21% the length of the total encoding, 60%, 80% and 95% of the encoding is sufficient to recover the I-, P- and B-frames respectively.

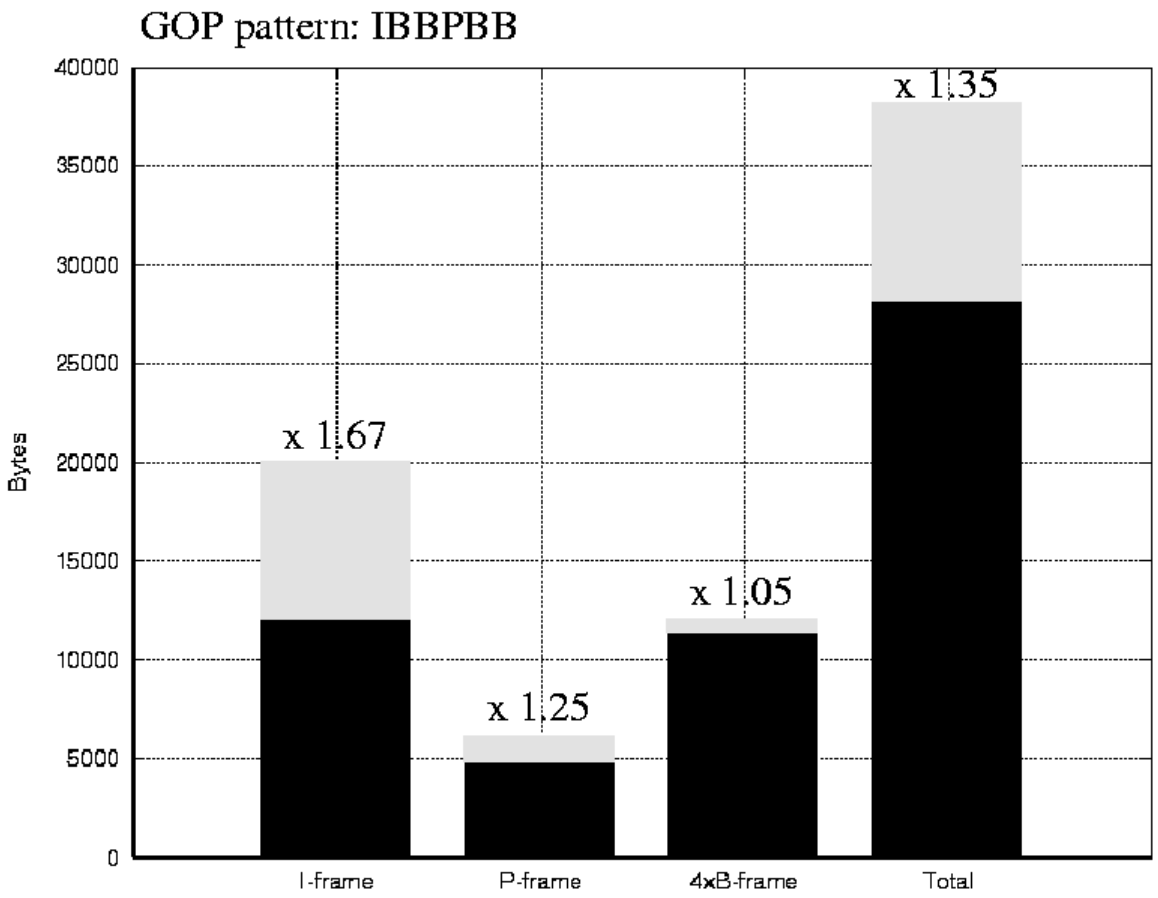

Figure 4.4. Multilevel redundancy distribution.

4.2.2 Priority Table Setup

The priority table is to be set up by the application. It should be structured as simple as possible in order to simplify the communication process with the PET interface. There are three major features, the interface has to know about:

- Number of priority segments

- Length of each segment
- Priority of each segment

A fixed number of 10 segments per message was considered to be enough for most MPEG groups of pictures. Each segment consists of either only header information, a single frame or a block of frames.

Priorities are inserted by fraction of packets needed to recover, multiplied by 1000. Consequently, 750 correspond to 75%. Returning to the example pattern, the priority table looks as follows:

Segment Size	12000	5700	4800	5700	Not used	Not used	Not used	Not used	Not used	Not used
Priority	600	950	800	950	/	/	/	/	/	/

Table 4.2. Priority table format.

Note that the groups of two consecutive B-frames form one priority segment. In total 20 values are passed in a zig-zag ordering (arrow) to the interface, even though in this case only eight are needed.

With respect to mapping the group of pictures onto packets according to the priority table, the following artificial scenario is considered:

- Packet size 190 words (380 bytes).
- Total number of packets sent: 100;

Packets might be considered to be UDP packets on top of an IP protocol. The total amount of packets sent is determined by the quotient of the total encoding of 38,000 bytes (Table 4.1) and the packet size. In order to meet the priorities specified in Table 4.2, the I-frame segment has to be encoded such that it can be reconstructed by any 60 packets (60% of the total amount of packets sent). Therefore the segments have to be divided into blocks of size 60 (polynomials of degree 59) for the I-frame, 80, and 95 for the other segments respectively. Figure 4.5 shows the striping procedure for the given example and the partitioning of. Note that this works exactly only for the given values in this example.

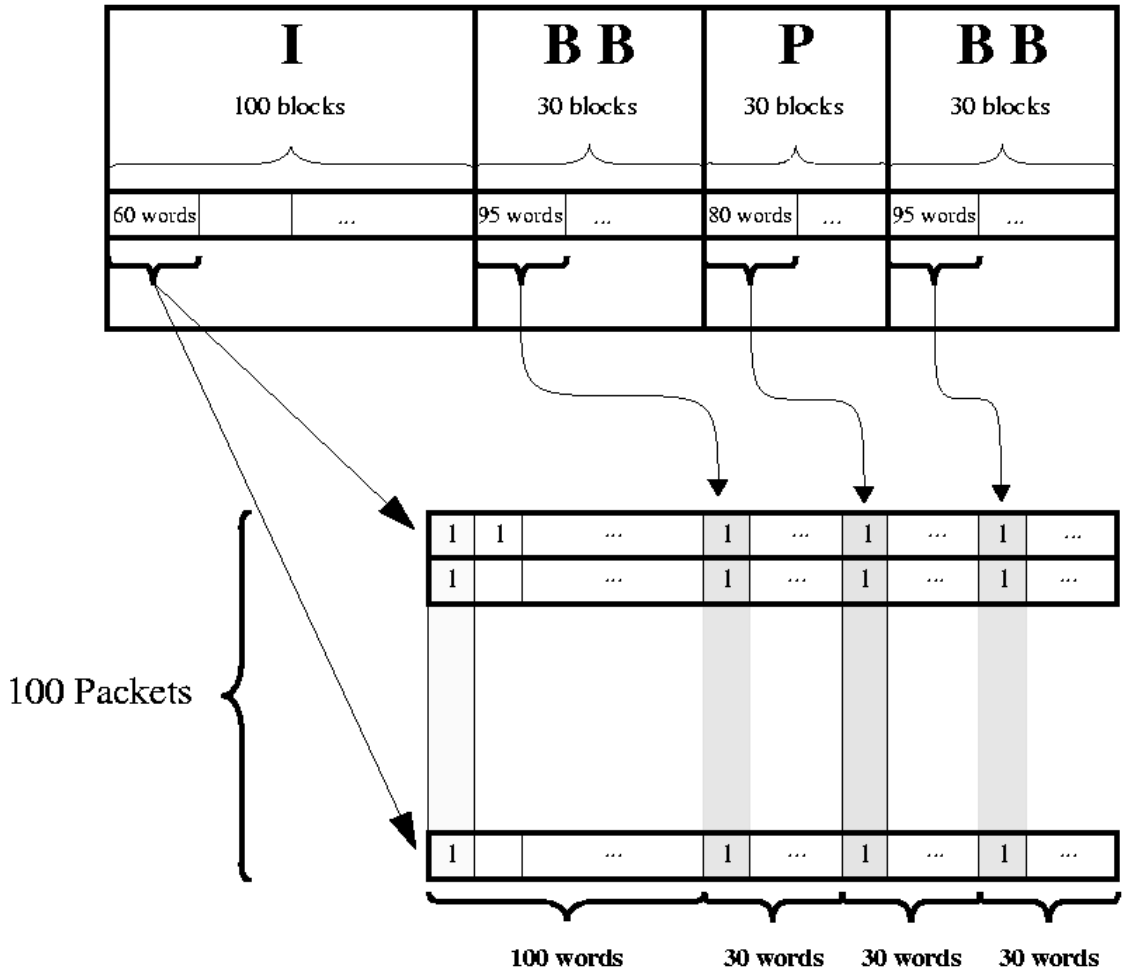

Figure 4.5. Striping process.

4.3 Software Architecture

Since Priority Encoding Transmission is designed to be compatible with any scalable application or type of network, it should be implemented as an easily exchangeable interface. An architecture in which the interface consists of two modules was introduced [Leicher, 94]: an *application module* handles the video data and passes a whole message at a time together with the priority table to a *PET coding module*. The latter creates the corresponding amount of packets needed and maps the message. On the decoding side, the PET coding module collects the transmitted packets, recovers the information and passes the video data to the application module (Figure 4.6).

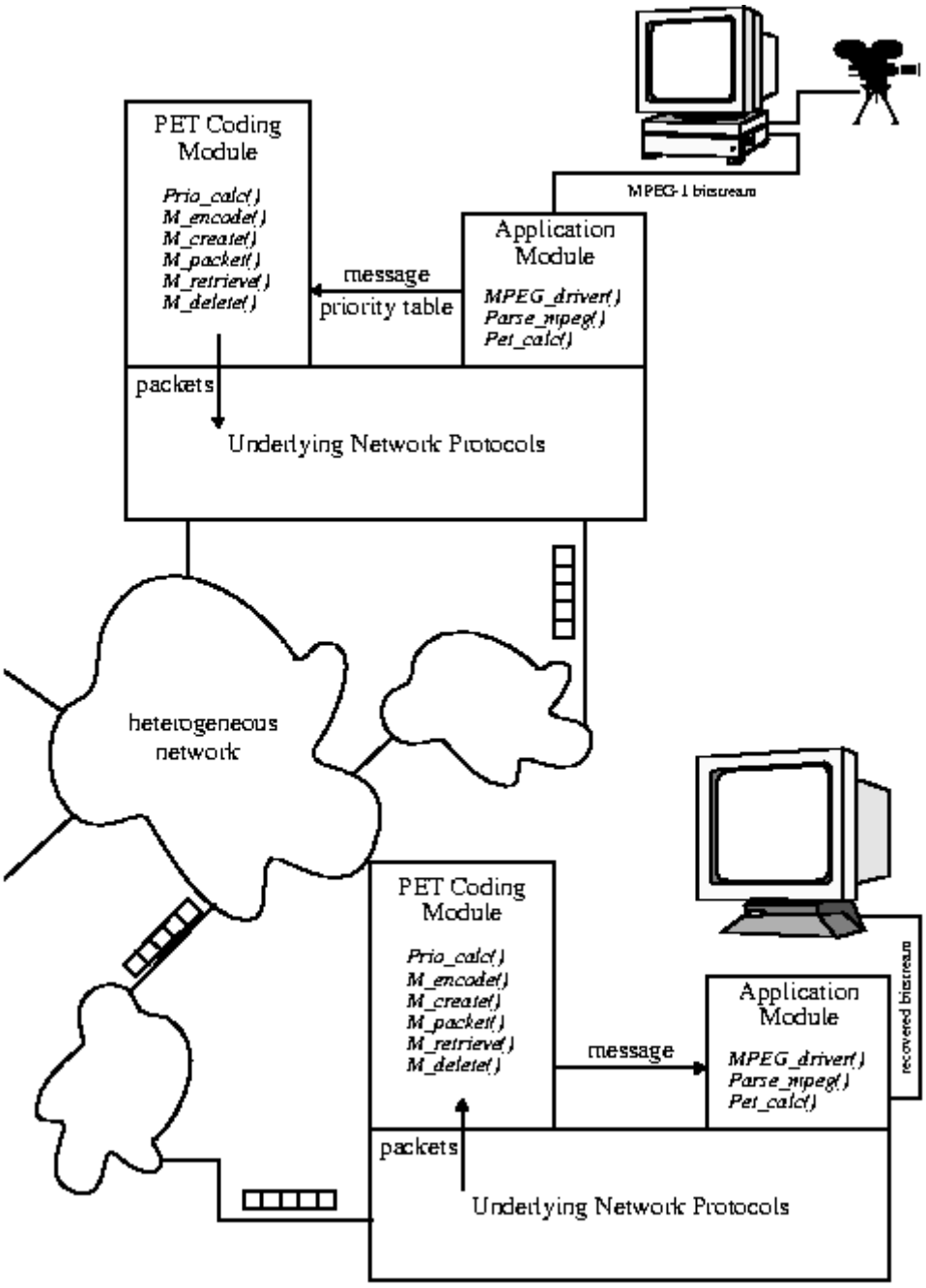

Figure 4.6. System Architecture.

The application module consists of the following routines:

- *MPEG_driver()* identifies the bitstream as an MPEG sequence and provides the user interface. The user might initialize the priority settings for the different frame types and header information. The size of the priority table could also be customized. According to the

picture size and GOP length it allocates memory space for the message buffer. All other routines are invoked by MPEG_driver().

- *Parse_mpeg()* parses the hierarchical bitstream, while reading a GOP into the message buffer. It decodes the different priority types and allocates a linked list that stores relevant data such as size, position in the buffer and number of frames about each segment.
- *Pet_calc()*. After a message is buffered MPEG_driver() calls Pet_calc(). According to the user specified priorities it fills in the priority table as shown earlier in Table 4.2. It also has to make sure that the message is word aligned, since this is the basic processing unit of the coding routines. Therefore in case a segment boundary is not word aligned it performs a zero byte stuffing.

The message and table are then passed to the PET coding module that encompasses the following routines:

- *Prio_calc()*. With respect to Figure 4.5 a kind of "internal priority table" has to be determined by Prio_calc(). Its input is the user-specified priority table, packet size and message size. The "internal table" has to have the size of a packet. Returning to the previous example its first value then would be 60, the size of the first message block and degree of the corresponding polynomial. How to fit in these values exactly into a packet and how many packets to send for a message will be discussed later on.
- *M_encode()* then performs the actual encoding according to the internal table over the Galois field GF[2^{16}+1]. It evaluates the polynomials and determines the offset. Up to now packet sizes are assumed of a couple of a hundred bytes, so the priority table is transmitted uncoded with each packet. Additionally the Galois field element and offset contribute to the total overhead (Figure 3.4).

These are all the routines needed on the encoding side. From the destination point of view the focus is on the recovery aspect:

- *M_create()* allocates memory for the message to be reconstructed and initializes the interpolation routines. This space later will be erased by invoking *M_delete()*.
- *M_packet()*. Each arriving packet is passed to M_packet(). Depending on the fraction of packets received it tries to regain the original information by interpolating the polynomials. In the current version the granularity of recovering is limited to priority segments, i.e., at least 60% of all packets are needed to recover some useful data.
- *M_retrieve()* compares the amount of received packets with the priority table and accordingly erases undecodable data. For simulation reasons it also calls another routine,

- *create_dummy_b()*, which replaces lost frames with dummy B-frames in order to keep the same number of images in the bitstream.

4.4 Simulation experiments

Primarily, PET has been applied to perform simulation and analysis of the resilience of an MPEG stream PET encoded. In a particular sample stream, a side-by-side comparison of MPEG stream encoding with and without PET demonstrates the dramatic improvement of picture quality due to PET, using only 24% redundancy on top of the standard MPEG data. For this particular group of pictures and frame sizes, this represents a more than 5 fold reduction in transmission rate over the JPEG mode to achieve comparable quality.

For the particular example stream, priorities were assigned in a way that header information can be recovered from any 10% of the encoded packets, I frames from any 60%, P frames from any 75% and B frames from any 90%. Computing the redundancy distribution:

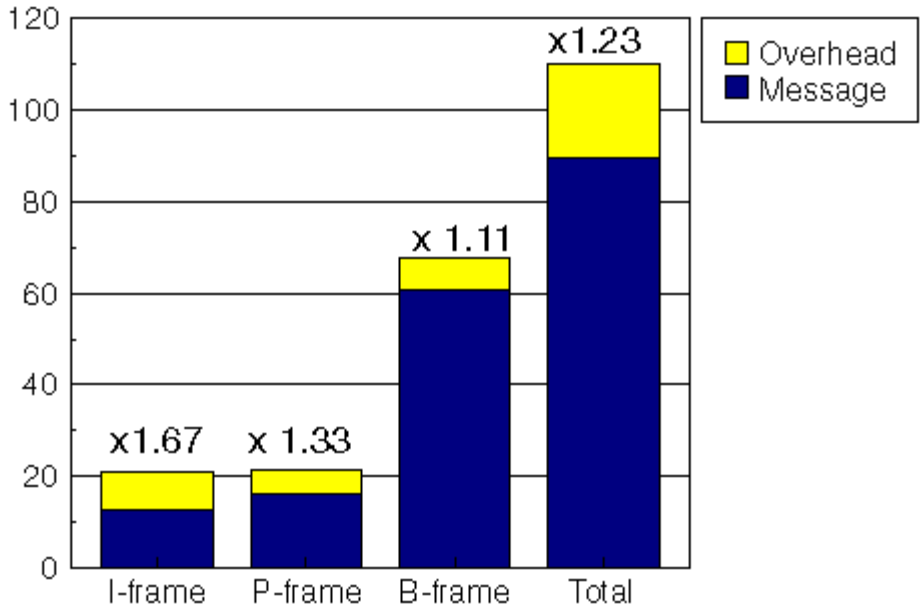

Figure 4.7. Redundancy distribution.

A total overhead of 23% enables the I-frame to sustain losses of 40%. Respectively P-frames sustain losses up to 25%, B-frames up to 10% and important header information up to 90%.

In the following two demonstration MPEG streams, encoded packets were randomly thrown away in order to simulate a lossy medium. The following viewgraph shows the number of packets

sent and received per message. The MPEG sequence "Red's Nightmare" was encoded into 41 messages, encoding one whole GOP at a time. A GOP consists of 1 I frame, 2 P frames and 27 B frames. The whole sequence encompasses 1210 frames and the packet size was set to 2000 bytes.

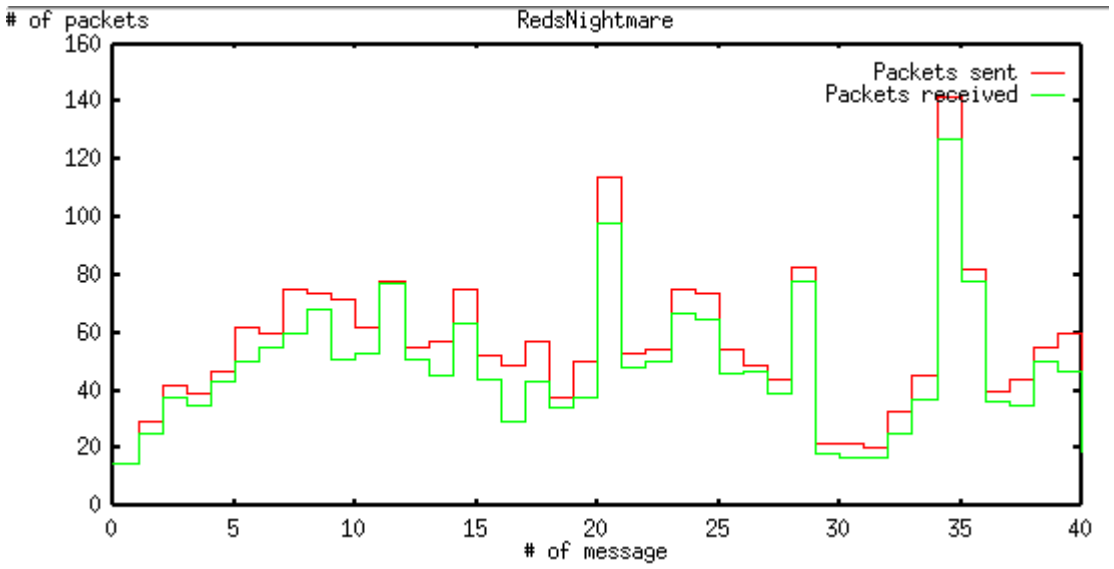

Figure 4.8. Packet loss statistics.

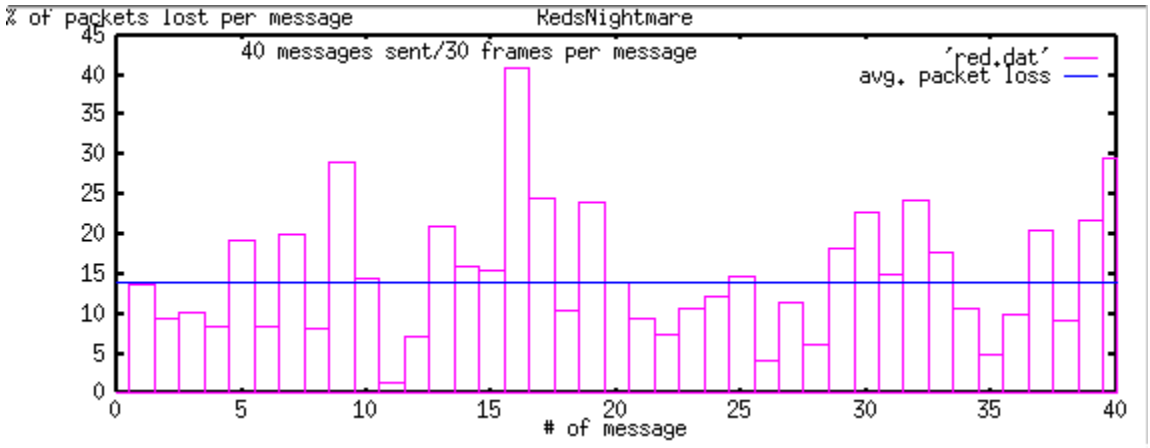

Figure 4.9. Packet loss statistics.

On the destination side the received packets were decoded and missing frames were substituted by "dummy" B frames that repeat the last correctly decoded frame. MPEG clips demonstrating the results can be found in http://www.icsi.berkeley.edu/PET.

5 PET implementation

5.1 Striping process

Given a message consisting of message parts with the 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 is 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 [Lamparter and Kalfane, 95].

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 round-off 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 2 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 4 bytes times the dimension of the finite field as an algebra over GF.

The larger the segment size, the larger the round-off 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 round-off 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, the number of segments needed for the lowest priority level first is computed. Then it is determined how much extra information could be sent without using any more segments, and that amount of information is sent 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. In fact, this choice does not hurt the PET guarantee, as it means that the information will be recovered from fewer packets than the user had asked for. Then the size of the second message part is decreased by the amount that has been sent at the lower priority level, and it is kept on doing so for each priority level. Hence, for all but the last priority level, full use of each segment of the packets is made, by sending less important information at a lower priority level than required. This helps us to lower the round-off 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 round-off 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:

$$N = \left\lceil \frac{E}{Sl} \right\rceil;$$

2. Starting at:

$$N = \left\lceil \frac{E}{Sl} \right\rceil;$$

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:

$$m_i = \lfloor P_i N \rfloor;$$

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:

$$s_i = \left\lceil \frac{L_i}{m_i} \right\rceil;$$

iii. The total amount of information that can be sent with this number of segment is s_i x n_i, so send (s_i x n_i)-L_i info from the next level, and decrease L_{i+1} by this amount. If i=k, i.e. if it is 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.

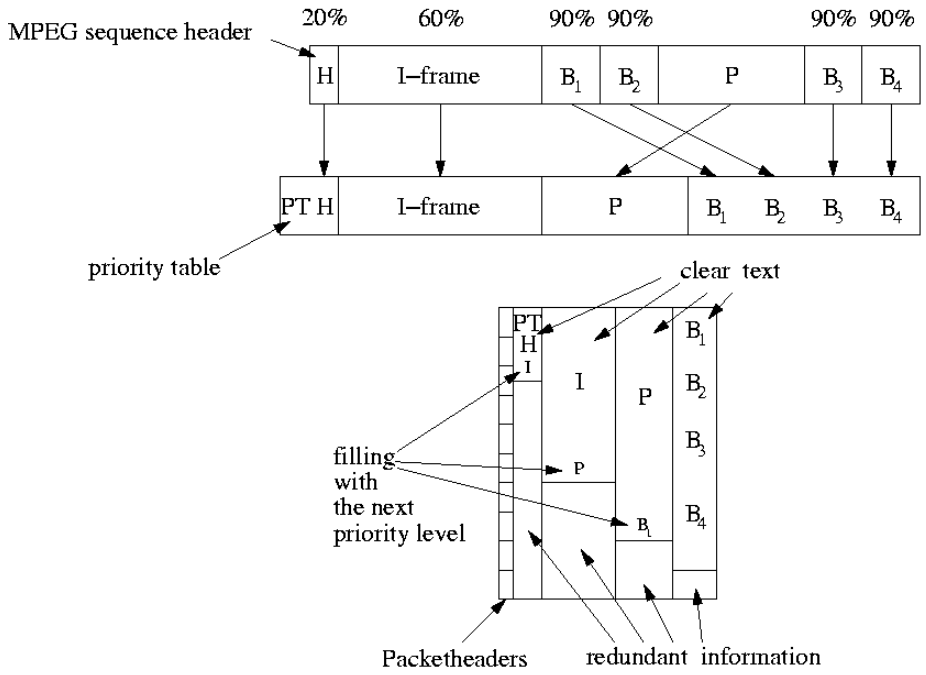

Figure 5.1. Sorting and striping of MPEG messages.

5.2 Packet format

The format of each packet follows the striping processes described in the previous paragraph [Lamparter and Kalfane, 95]. Each PET packet (see fig. 5.2) 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.

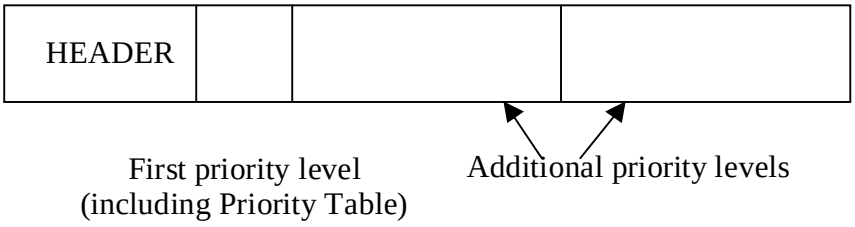

Figure 5.2. PET packet format.

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, sometimes it is 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, the priority table is found as clear text 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 Figure 5.2):

- *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 sent. Because there are only 256 different Ids available, the protocol between sender and receiver must ensure that the receiver has never an old message with the same ID in his memory. In rare case of very high losses it may occur that the receiver gets confused by two messages 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 receiver 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. This and the next two values are needed to extract the priority table;
- *pp_tab*. The priority of the priority table;
- *priol*. The length of the priority table;

5.3 Data Structure

There are two basic data structures in PET: the priority table (PETpriorityTable) and the result table (PETresultTable).

The priority table is used during the encoding stage to store information such as the lengths of the different message parts and their corresponding priorities. The information in the priority table is used during the decoding stage by the receiver to compute how the message parts are to be recovered from the received packets. The priority table is itself encoded into the packets, and is decoded on the receiving host before the message parts can be recovered. The result table is used only during decoding to get exact information about which message parts are recoverable and which message parts the application is asking PET to recover.

5.3.1 The Priority Table

An entry of the table consists of the following fields:

- **int prio**. Defines the priority. The message part can be completely recovered, if at least prio% of the packets arrive at the receiver.
- **int length**. The length of this part in bytes. The maximum length is defined as PET_MAX_PRIO_LENGTH. In the current version this value is 2^{32}-1=4.294.967.295.
- **char type**. The application may send one byte of descriptive information for each message part in the priority table. In MPEG this is used to transmit the frame type.
- **u_char *data**. A pointer to the data field of this message part (for encoding purposes only).

The following fields are used only at the receiving side. PET updates this field each time it receives a new packet:

- **int minpackets**. Number of packets needed to recover this part completely;
- **int recoverable**. True if this part can be recovered;
- **int cheapRecover**. True if the recover process is cheap, i.e., the flag is true when PET can decode this message part without evaluating redundant packets.
- **int recoverTime**. PET calculates a prediction of the CPU-time needed to recover this part. The time is given in μsec (not yet available feature).

5.3.2 The Result Table

This table is used when the application decides to ask PET to retrieve the message (or parts there of). The application allocates memory for this table and fills in much of the information, although

PET fills in some of the information. The structure of this table must match the structure of the corresponding priority table. Each entry consists of the following fields:

- **int recover**. Set to true by the application if it wants PET to recover this message part.
- **int valid**. Set to true by PET if this part was recovered completely.
- **int length**. The length of this message part in bytes[5].
- **char type**. The application provided descriptive byte of information[6].
- **char * data**. The data of the message part (Set by PET, pointing into a application provided piece of memory).

Some applications may want to evaluate portions of a message even if the message part wasn't recovered completely. The following fields deal with this issue. They are only valid if the part wasn't recovered completely, i.e. the valid field is false. They contain exact information of valid and invalid portions of the message part. All invalid portions are set to zero. Unlike the rest of the result table, PET allocates memory for this additional portion of the result table and fills in the information.

- **int Lsegs**. Size of a segment in bytes. Segments are the internal units with which PET deals.
- **int Nsegs**. Number of segments in this part (partial and complete segments).
- **int Ngoodsegs**. Number of recovered segments.
- **int offset**. The message part usually starts partway into the first segment that contains information about the part. This is the offset into the first segment where the message part starts (see Figure 5.3).
- **int *RecIndex**. An array with a boolean value for each segment. The value is true, if the segment was recovered, false otherwise. Attention: These values are undefined after destroying the corresponding message. A given byte j in a message part is valid if and only if RecIndex [(j+offset)/Lsegs] is true.

[5] The length and type field is a copy of the ones in PETpriorityTable. They were added for the convenience of the application programmer.

[6] The length and type field is a copy of the ones in PETpriorityTable. They were added for the convenience of the application programmer.

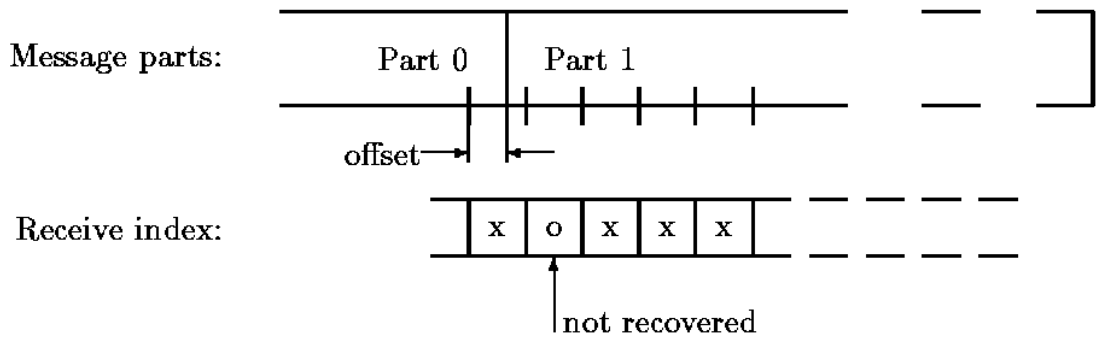

Figure 5.3. Partial losses of message parts.

5.3.3 PETmessageID

The PETmessageID is a simple integer variable in the range 0..255. It is used to distinguish between several messages arriving from one sender. This sender is responsible for choosing appropriate ID's. If there are more than one sender, it is important that the receiver opens a unique PETdecoder for each sender. If two senders are using the same PETmessageID at the same time, PET will get confused.

5.4 PET encoding

There are only 2 functions for the encoding of a message:

- **PETnumberOfPackets**
- **PETencode**.

5.4.1 Calculation of the packet number

The application has to allocate memory for the packets to be later filled in by PET. In order to know how many packets are needed for the given message:

int PETnumberOfPackets(PETpriorityTable *, int prioLength,
int priopriotab, int *packetLength)

The meaning of the parameters is as follows:

- int PETnumberOfPackets(…) returns the number of packets needed to encode the given message;
- PETpriorityTable* pt. A Priority Table (TP) as described in Section 5.1;

- int prioLength. Length of the priority table;
- int priopriotab. The priority of the priority table. The priority table is encoded in the packets in the same way as the message parts. PET insures that the actual priority of the priority table is set at least as high as the priority of any message part. For example, the application may set the value to 100 (i. e., no protection) and then PET would automatically re-adjust this priority to the highest level specified for the message parts;
- int *packetLength. The application gives PET the maximum length of a packet. Because of the internal organization of the data (segmentation) sometimes PET cannot use the full length and will return the used length of the packet here.

PETnumberOfPackets() uses only the fields length and prio of the priority table.

5.4.2 Calculation of the packet number

After allocating the memory for the packets and setting the data pointers in the priority table, the application may use PETencode() to encode the packets:

```
int PETencode(      PETpriorityTable* pt, int prioLength, int priopriotab,
                    int MessageID, Number **packets, int Npackets,
                    int packetLength      )
```

- int PETencode() returns true if no error was found and the encoding was done correctly, otherwise it returns false. Possible errors are: Message part too long, an invalid priority, or an incorrect number of packets;
- PETpriorityTable* pt. The priority table should be the same as in the call to PETnumberOfPackets(), but additionally the data pointers must be filled in by the application and the application may also fill in the type field;
- int prioLength. Same as above;
- int priopriotab. Same as above;
- int MessageID. The sender is responsible for assigning a unique id to each message. If there are only a few messages active at any point in time, the application may cycle through a limited number of ids. If many messages are active at once, the application may require a larger range of message ids. A typical solution is for the sender to cycle through the message ids 0 through 255;

- Number **packets is an array of pointers to memory for the packets. The memory must be aligned to multiples of four bytes;
- int Npackets is the number of packets in the array of packets. PET checks if this number corresponds to the number of packets needed to encode the given message.

5.5 Typical sender behavior

Usually the sender loop looks like as follows:

```
MessageID := 0
forever
        collect data for the message
        set priorities and lengths in the priority table
        call PETnumberOfPackets()
        allocate memory for the packets
        set data pointers and type fields
        call PETencode()
        send the packets
        free the packets
        MessageID := (MessageID+1) % 256
```

5.6 PET decoding

The process of decoding is somewhat more complicated. The receiver has to collect the packets, hand them over to PET, and at some point decide to decode the message. The application also has to distinguish between several senders.

5.6.1 Opening a PETdecoder

To take care of several senders, the application opens a PETdecoder for each sender:

PETdecoder *PETopenDecoder(unsigned int flags, PEToption *po)

PETopenDecoder() uses the same structure to initialize as PETinit(). Unset values are filled with the value given in PETinit(). It is permitted to use the NULL pointer. The function returns a pointer that is used in calls to decode routines of PET.

5.6.2 Processing Received Packets

For each packet received the application calls a processing routine:

```
PETmessageID PETprocessPacket(    PETdecoder *pd, Byte *packet,
                                  int length, PETpriorityTable** pt,
                                  int *prioLength, int *MessageSize    )
```

- PETmessageID PETprocessPacket(…). The return value is the message id of the processed packet. From this information the application can tell whether the packet belongs to an old message or to a new one.
- PETdecoder *pd. The sender id created by PETopenDecoder() for each sender.
- Byte *packet. The packet PET should process. PET copies the packet into its own memory area so the memory may be reused.
- int length. Length of the packet. PET checks if all packets have the same length.
- PETpriorityTable** pt. PET puts a pointer to the priority table at this address. After destroying the message this pointer is not longer valid (see remarks for MessageSize).
- int *prioLength. PET writes the length of the priority table at this address.
- int *MessageSize. PET writes the total length of the message at this address. The length includes the internal overhead used by PET for the decoding of the priority table.

The three values are only set if three conditions hold:

1. pt is not the NULL pointer,
2. prioLength is not the NULL pointer,
3. PET has enough packets to decode the priority table.

5.6.3 Retrieving the message

The application has to decide when it has enough packets to decode the packets that have arrived. It may use timing for this purpose or simply start decoding of the current message when the first packet of the next message arrives. The application may call PETretrieveMessage() several times

for the same message, so it can retrieve the important parts earlier than less important ones. In a multithreaded environment the routine may be invoked several times, each time on a different set of message parts, for parallel retrieval of the entire message. PET has been designed with coordination mechanisms that ensure inconsistencies do not arise in a multithreaded environment. Before calling PETretrieveMessage() the application has to allocate a buffer for the message to be retrieved. PET assigns a pointer to each decoded message part. Those memory areas are in the buffer allocated by the application. Hence the application may free this buffer whenever it no longer needs the message parts. In addition, the application has to allocate memory for the result table and fill in values to indicate which message parts are to be retrieved.

int PETretrieveMessage(PETdecoder *pd, PETmessageID message,
Byte *buffer, PETresultTable *outputTable,
int prioLength)

- int PETretrieveMessage() returns true if no error occurred.
- PETdecoder *pd. The sender id for the PETdecoder.
- PETmessageID message. The message id as returned by PETprocessPacket().
- Byte *buffer. This is a buffer space used by PET to decode the message. The message parts are located in this area. The application has to provide this buffer with at least the size assigned to MessageSize in PETprocessPacket().
- PETresultTable *outputTable. This table is used in two ways: First the application specifies which parts of the message should be decoded, and second PET describes the result of the decoding process. In contrast to the priority table, the application is responsible for allocating the memory of the result table. It should have entries corresponding to the entries in the priority table.
- int prioLength. The application has to tell PET the length of the result table so PET can compare it with the priority table PET created.

5.6.4 Destroying a message and closing the decoder

After retrieving a message and evaluating the result the application should have PET destroy all information PET created associated with the message, i. e., the packets, the priority table and portions of the result table originally allocated by PET. This call deletes all the memory created

and used by PET during processing of the message. Note: The receive index (RecIndex) in the result table is also deleted.

PETmessageDestroy(PETdecoder *ph, PETmessageID message)

- void PETmessageDestroy(). Destroys all memory associated with the given message created and used by PET.
- PETdecoder *ph. The decoder.
- PETmessageID message. The id of the message.

void PETclose(PETdecoder *ph). Closes the decoder for the given handler. This function should be called when the according incoming data stream is closed.

5.6.5 Miscellaneous functions

The following two functions allow the application to ask the PET decoder for some internal values:
int PETrecPackets() returns the number of received packets while int PETnumPackets() returns the number of the total amount of expected packets. Both functions have the same parameters (xxx stands for rec and num):

int PETxxxPackets(PETdecoder *ph, PETmessageID id)

- PETdecoder *ph. The decoder.
- PETmessageID id. The id of the message.

If the message id is invalid then the functions return a PET error code.

5.6.6 Typical Behavior for Receiving Messages

A simple version of the loop for the receiver is the following (it is assumed there is only one sender active):

```
forever
        receive packet
        newid := PETprocessPacket()
        if newid != id
```

allocate the buffer for the message
call PETretrieveMessage
evaluate message
delete memory for the message
call PETdestroyMessage
id := newid

6. MPET in VIC

VIC is a popular application for interactive video developed at UCB/LBL. VIC stands for video interactive conferencing [McCanne, 95]. It consists of several encoding schemes (NV, h.261, JPEG, CuSeeMe, etc.) and runs on workstations and PC of all major vendors. The PC version has been ported and improved at UCL[7].

6.1 MPEG decoding

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

The complete player was translated from C to C++ and all global variables became class members of the MPEG decoder class. The old program 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 consumer 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.

6.2 MPEG encoding

For the encoding the SUNVIDEO board has been used. Through the GUI of VIC (Figure 6.1) the user can specify the desired frame and bit rate. An additional window (Figure 6.2) allows the user to set the pattern of the MPEG frames and various PET parameters.
There is the possibility to choose the source between either an MPEG file or a *live* camera through XIL.

[7] http://www-mice.cs.ucl.ac.uk/multimedia/

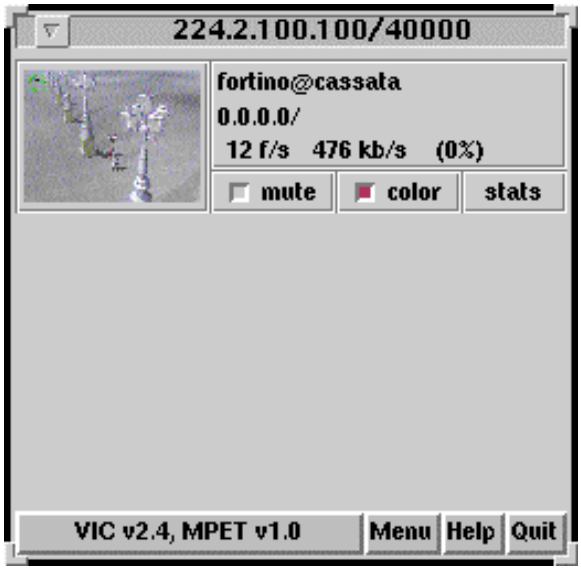

Figure 6.1. GUI of VIC-MPET.

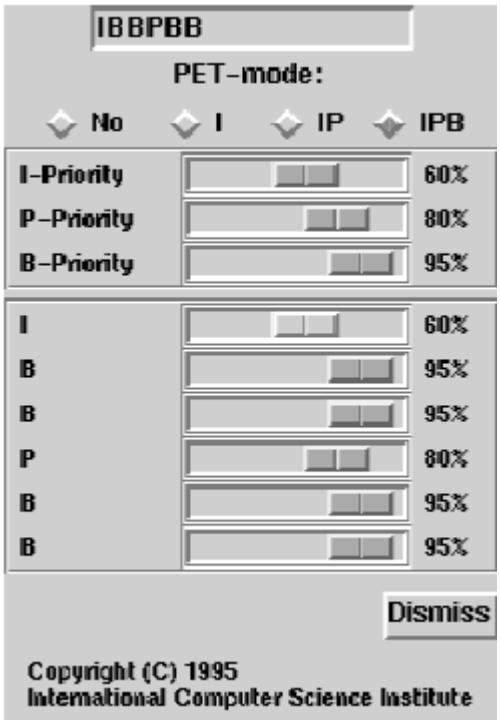

Figure 6.2. The MPET window.

6.3 RTP encapsulation of plain MPEG streams

6.3.1 Header fields

The MPEG packets are RTP encapsulated [Schulzrinne et al., 96], but currently uses none of the MPEG specific header fields. It is planned to support RTP-encapsulation as defined in [Hoffman et al., 98]. Currently the MPEG specific header is four bytes long, *non* is set, and *all* ignored (see §2).

6.3.2 Payload

According to [Hoffman et al., 98] 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 re-assembler does not expect the packets to have equal length.

6.3.3 Loss recovery and handling of late packets

Losses are handled differently according to the type of the affected frame. Hit 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 (i.e., it did not find the end marking of a macro block), it 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, it might be a sign that the decoder is too slow to keep up with the incoming data stream. Eventually that behavior 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 hit B-frames.

6.4 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 [Schulzirinne at al., 96]. The subtype is additional information on the exact encoding used. In MPEG encoding three different schemes are currently used:

- **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.

6.4.1 Timing: when to play a frame

Figure 6.3 shows the timing for a MPEG stream with the frame pattern IBBPBB.

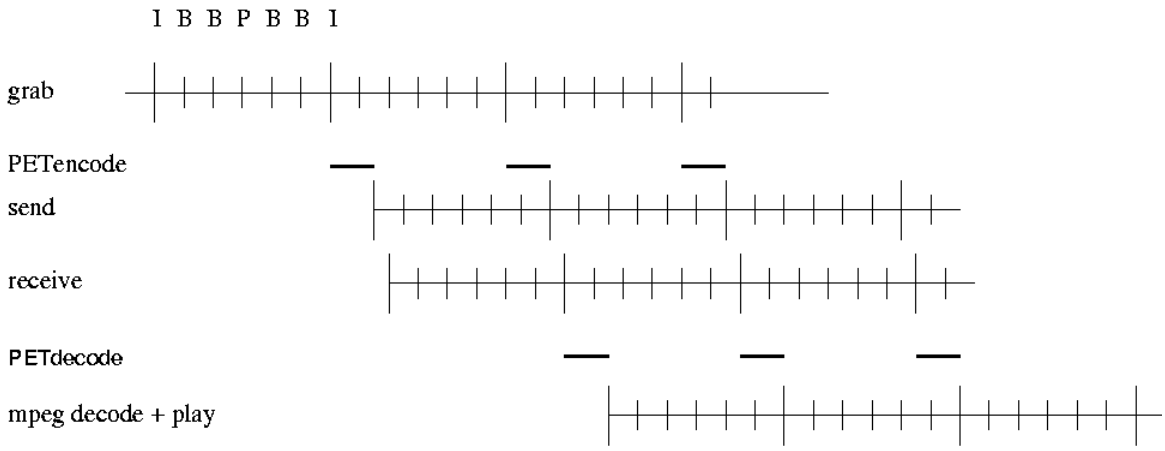

Figure 6.3. Timing of MPEG frames.

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, the next frame will not miss to be grabbed. 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 (unfortunately the XIL-library refuses to give us the real capture time. So it is picked-up by the operating system). 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 it is checked the correct time schedule. In case a delay occurred and if there is a B-frame to decode, the frame is discarded and the Tcl-timer restarted. If a P-frame has to be managed and there is only one frame behind, the frame is decoded and displayed. If there is an I-frame, it is decoded and displayed in any case.

With this scheme the overloading of the CPU, which would result in a loss of packets in the incoming buffer, is avoided. 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. It should be made sure that the buffer is emptied frequently. The simplest way would be, to give the corresponding Tcl-routine a high priority. Unfortunately, the Tcl event mechanism has no priority scheme. Hence it should be made sure, that the timing mechanism leaves at least a few milliseconds till the next frame is due. The Tcl scheduler calls the packet reader in this gap and empties the buffer.

6.4.2 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 at 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.

A possibility to know whether or not the frames are behind the schedule is given by taking a look at the buffer length. After reading a packet from the queue it is checked 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.

6.4.3 Parallelizing PET operations by using multiple threads

Many modern UNIX variants have support for multiple threads. These 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 its data.
In addition PET itself may take advantage of multiple processors in a machine. It may compute priority levels or packets in parallel.

Protection against reentry

If several threads manipulate the same set of data, inconsistencies may occur. Therefore it is needed 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 concurrency 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 semaphores are used to signal the end of a thread to several other threads.

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

Compute the priority levels in parallel

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

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

On a processor machine this computation would take 21s. But what is happening on our 4 processor machine? The first 1.5s all 6 tasks are running on 4 CPUs, then it runs another 1.2 s with 5 threads, and after another second one CPU gets out of usage. The last second only one CPU is busy. That gives a speedup of only 21/6.7 ≈ 3. A better scheduling of the tasks would help, as long as the number of task is as large as the number of CPUs. In the example a speedup of 21/6 = 3.5 has been obtained. The implementation of the parallelization of the different priorities is straightforward. But on the decoding side, things are more complicated. The application may start several decoding threads where it wants to have different message parts. It 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 the thread is started
                        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 the waiting incarnations of PETretrieveMessage() are restarted.

Compute the packets of one priority level in parallel

To encode a priority level two types of packets need to be produced: in a first step the message is copied into the clear text packets and in the second the redundant packets are computed. 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.

Further considerations

More experiments need to be done 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 (≈50µs according to the handbook), but there is a cheaper possibility: it is a client/server model with multiple servers for the encoding. A dispatcher is then responsible to hand the encoding task to the next free server thread and a mechanism is needed 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.

7. Conclusions and future work

PET can be used with any packet-switching network hardware or protocol. It works as a layer above the transmission of packets (ATM cells, etc.). By using coding techniques, PET improves the quality of transmission over packet networks that have unpredictable losses. PET uses a unique multilevel forward error correction (FEC) scheme [Albanese et al., 97] which introduces a minimum increase in the transmission rate. PET makes it possible MPEG transmission over Internet, which has not been practical due to bursty losses. PET has been also embedded into a version of the video-conferencing tool VIC to protect the real-time transmission of MPEG streams, and this worked successfully.

7.1 Future Work on FEC codes

The very first implementation of FEC codes that was used in PET were not fast enough even for encoding and decoding moderate sized files. As reported previously, this led to the invention of some fairly fast FEC codes for PET implementations. Although these codes are reasonably fast for moderate sized files, they are not adequate for real-time encoding and decoding of large files, as the time scale quadratically with the size of the file. As reported earlier, some faster FEC codes were invented and pointed out in [Alon and Luby, 96].

7.1.1 Tornado codes

The potential for FEC codes in networking applications has not yet been fully realized, primarily because previous software implementations of FEC codes have been too slow. Their running times are only reasonable when the number of redundant packets is small, e.g., at most 100 packets. The reason for this slowness is that the encoding and decoding time of standard FEC codes are proportional to the length of the encoding times the number of redundant packets. Other known codes based on the Fast Fourier transform have asymptotically much better decoding times, but in practice their complexity also makes them too slow to decode large messages (there are implementations of such codes in custom designed hardware that run quite fast).

Over the past two years, a completely new class of codes has been implemented, called *Tornado codes*, which promise to make the FEC solution practical [Luby et al., 97]. Tornado codes have the property that *slightly more* than the ideal number of encoding packets must be received to decode the message: it is called the per cent needed over the ideal number the **length overhead**

(recall that the ideal number is the original number of message packets). By allowing some length overhead, much faster coding schemes can be developed. Tornado codes can be encoded and decoded in time proportional to the length of the encoding times a small constant that is independent of the number of redundant packets. This small constant, which is called the **time overhead**, is typically around 6.

These codes were implemented in software, and the results are very encouraging. As an example, a 32M bytes file was took and partitioned it into 64K message packets of 512 bytes each. Using Tornado codes, 128K encoding packets from these 64K message packets were produced. 10,000 trials were run, where each trial consisted of randomly choosing and receiving encoding packets until there are enough to decode all of the original 32M byte file. In these 10,000 trials, the average length overhead was only 3.2%; in other words, about 66K encoding packets were enough to decode. Furthermore, in less than 1% of the trials the length overhead was more than 4.0%.

Although there is a modest length overhead using Tornado codes, the advantage is that the encoding and decoding times are very small. On a DEC Alpha machine, the time to produce the 128K encoding packets from the 64K message packets is just under 2s, and the time to decode the 32M byte file from 66K encoding packets is also just under two seconds. The encoding and decoding times for a message and encoding of this size using standard FEC codes would be at least 10,000 times slower, i.e., tens of hours. Thus, the advent of Tornado codes makes software solutions possible for problems that are orders of magnitude larger than were previously conceivable.

It worth emphasizing that Tornado codes have a number of important features:

- **Simplicity:** Tornado codes use only simple XOR operations, and no complicated data structures are required for their implementation.
- **Generality:** techniques were developed to design Tornado codes for whatever rate, or redundancy level, is specified. (In a rate 4/5 code, for example, 4/5 of the encoding is the original message, and 1/5 of the encoding is redundancy.) Moreover, codes were designed to decrease the length overhead at the cost of a small increase in the time overhead.
- **Provability:** a mathematical formulation of Tornado codes was developed that allows us to prove how well they will perform. Furthermore, an infinite family of codes was found for which it can be proved that the length overhead decreases very quickly as a function of a growing time overhead. In [Luby et al., 98], a new analysis of Tornado codes that can be used to analyze a number of other problems as well has been found.

- **Applicability:** Tornado codes can be used to encode and decode large files in real-time. This makes Tornado codes attractive to use for a number of networking applications.

Over the next year, it has been planned to aggressively pursue the incorporation of Tornado codes into networking protocols. A multicast distribution protocol written in JAVA that uses Tornado codes has been implemented. Preliminary tests of this nascent protocol have been very encouraging.

7.2 Future Work on Applications for PET

Currently the prioritizing is used only for MPEG GOP coding. Another possible scheme can prioritize the DCT matrix coefficients according to their level of importance [Storn, 95]. 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, on the other hand, 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. However, 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 a 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, e.g., the 3-D subband coding [Taubman and Zakhor, 93]. The layer can easily be mapped to the PET priority levels.

7.3 Integration of PET with MASH

MASH Toolkit [McCanne et al., 97] is an outgrowth of the Internet MBone tools. It is based on IETF standards including IP-multicast, RTP, RTSP, SIP, and SAP. In addition, it supports application-level protocols, such as SRM and SNAP, being developed to support flexible collaboration and media processing. MASH is going to be the platform [Rowe, 99] on which researchers will be able to experiment with computer-based conferencing and collaboration. A contribution to this initiative is to integrate PET with MASH[8]. PET is already implemented as a module. According to the MASH source/filter/sink abstraction it can be viewed as a filter object. For instance, a source is a video capture device with an MPEG encoder, the filter is the PET module, and the sink is a network transmission protocol (e.g., RTP). Thus, MASH has to be

[8] http://www-mash.cs.berkeley.edu/mash/

enhanced by introducing several new objects such as MPEG grabber through XIL, MPEG decoder, etc.

REFERENCES

[Albanese et al., 96] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan, "Priority Encoding Transmission," IEEE Transactions on Information Theory (special issue devoted to coding theory) Vol. 42, No. 6, November 1996.

[Albanese et al., 97] A. Albanese, M. Luby, J. Blomer, and J. Edmonds, "System for Packetizing Data Encoded Corresponding to Priority Levels where Reconstructed Data Corresponds to Fractionalized Priority Level and Received Fractionalized Packets," US Patent, N. 5617541, April 1997.

[Alon and Luby, 96] N. Alon and M. Luby, "A Linear Time Erasure-Resilient Code With Nearly Optimal Recovery," IEEE Transactions on Information Theory (special issue devoted to coding theory) Vol. 42, No. 6, November 1996.

[Blomer et al., 95] J. Blomer, M. Kalfane, R. Karp, M. Karpiski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," ICSI Technical Report TR-95-048 August 1995.

[Crowcroft et al., 99] J. Crowcroft, M. Handley, and I. Wakeman, "Internetworking Multimedia," UCL press, 1999.

[Hoffman et al., 98] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar, "RTP Payload Format for MPEG1/MPEG2 Video," IETF RFC 2250, January 1998.

[ISO11172] CCITT Recommendation MPEG-1; Coded Representation of Picture, Audio and Multimedia/Hypermedia Information; ISO/IEC 11172 Geneve Switzerland, 1993.

[Lamparter and Kalfane, 95] B. Lamparter and M. Kalfane, "The Implementation of PET," ICSI Technical Report TR-95-047 August 1995.

[Lamparter et al., 95] B. Lamparter, A. Albanese, M. Kalfane, and M. Luby, "PET - Priority Encoding Transmission: A New, Robust and Efficient Video Broadcast Technology," ICSI Technical Report TR-95-046 August 1995.

[Leicher, 94] C. Leicher, "Hierarchical Encoding of MPEG Sequences Using Priority Encoding Transmission (PET)," ICSI Technical Report TR-94-058 November 1994.

[Luby et al., 97] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical Loss-Resilient Codes," 29th ACM Symposium on Theory of Computing 1997.

[Luby et al., 98] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of Random Processes via Or-And Tree Evaluation," in Proc. of Symposium of Discrete Algorithms, 1998.

[McCanne et al., 97] S. McCanne, & others, "Toward a common infrastructure for multimedia-networking middleware," in Proc. of the 7th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97), St. Louis, Missouri, May 1997.

[McCanne and Jacobson, 95] S. McCanne and V. Jacobson, "vic: a flexible framework for packet video," In Proc. of ACM Multimedia '95, November 1995.

[Rowe, 99] L. Rowe, "Continuous Media Middleware Consortium," Proposal submitted to NSF, March 1999.

[Schulzrinne et al., 96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a Transport Protocol for Real-Time Applications", IETF RFC1889, January 1996.

[Storn, 95] R. Storn, "Modeling and Optimization of PET-Redundancy Assignment for MPEG Sequences," ICSI Technical Report TR-95-018 May 1995.

[Taubman and Zakhor, 93] D. Taubman and A. Zakhor, "Multi-rate 3-D subband coding of video," In IEEE Transactions on Image Processing, April 1993.