REIHE INFORMATIK
12/94

XMovie: Architecture and Implementation of a Distributed Movie System

R. Keller, W. Effelsberg und B. Lamparter
Universität Mannheim
Seminargebäude A5
D-68131 Mannheim

XMovie: Architecture and Implementation of a Distributed Movie System

Ralf Keller, Wolfgang Effelsberg and Bernd Lamparter
Praktische Informatik IV
University of Mannheim
e-mail: keller@pi4.informatik.uni-mannheim.de

Abstract

We describe a system for storing, transmitting and presenting digital movies in a computer network. The hardware used in the system is standard hardware, as found in typical workstations today; no special hardware is required. The movies are shown in windows of the X window system. This allows full integration with the classical components of computer applications such as text, color graphics, menus and icons. The XMovie system has several innovative features: First, it contains a new algorithm for the gradual adaptation of the color lookup table during the presentation of the movie to ensure optimal color quality on low-end workstations. Second, it is a multi-standard system supporting the compression techniques MPEG, H.261, Motion JPEG, and a newly developed extension to the well known Color Cell Compression method. Third, we introduce AdFEC, a new adaptable forward error correction method for our movie transmission protocol.

Categories and Subject Descriptors: C.2.2 [**Computer-Communications Networks**] Network Protocols – *protocol architecture*; C.2.4 [**Computer-Communications Networks**] Distributed Systems – *distributed applications*; E.4 [**Coding and Information Theory**] *data compaction and compression*; H.5.1 [**Information Interfaces and Presentation**] Multimedia Information Systems – *video (e.g., tape, disk, DVI)*; H.5.2 [**Information Interfaces and Presentation**] User Interfaces – *windowing systems*; I.4.2 [**Image Processing**] Compression (Coding) – *approximate methods*

General Terms: Algorithms

Additional Keywords and Phrases: distributed multimedia systems, digital video, transmission protocols, software motion picture

1 Introduction

Modern workstations have fast processors, high resolution color graphics displays and high-performance network adapters. For all these components considerable performance improvement can be still expected within coming years. Such a powerful hardware scenario enables new computer applications, of which the most important is perhaps the integration of digital films into a classical data processing environment.

For many years, computers have been used for the processing of digital films. The most important directions in research and development can be classified into the categories computer animation [35], computer simulation and digital video [37, 39]. This paper concentrates on the transmission and presentation of sequences of digital images in a distributed system of standard UNIX workstations and servers. We present an experimental system called XMovie which has been developed and is now operational at the University of Mannheim. The system allows the presentation of digital movies in a window of the X window system (X) under UNIX [53]. Unlike most other systems reported on in the literature, such as the Berkeley Continuous Media Player [51] or Pandora [23], the XMovie system requires no special hardware.

XMovie differs from other software solutions for digital movies (e.g. the Berkeley MPEG player [45], QuickTime [2] or Software Motion Pictures [42]) in that he has an *integrated movie transmission protocol*, thus enabling access to movies accross networks. Digital movies stored in a distributed system cannot be accessed online today. Even in the *World-Wide Web* (W3 [3]), a distributed hypermedia system, movies are transmitted via FTP, stored and then played back locally.

In Chapter 2 of this paper we describe the overall architecture of the XMovie system. In Chapter 3 we present formats, compression techniques and display methods for movies. Chapter 4 discusses application layer issues for movie transmission protocols. In Chapter 5 we concentrate on the implementation of XMovie, new developments and experiences. Chapter 6 concludes the paper.

2 Architecture of the XMovie System

The XMovie System is a distributed test bed for integrated transmission and presentation of digital movies. It consists of interconnected UNIX workstations, and it can transmit stored movies over digital high–speed networks and display them in windows of the X window system. Unlike other digital movie systems, it requires no special hardware in workstations. Relying on special hardware seems to be too constraining for innovative multimedia systems, even though it can help provide better performance in the short term. Our approach is to use software solutions wherever possible, and special hardware only when it is absolutely necessary (e.g. a video digitizer board as the source of a live movie). As an immediate consequence our system is highly portable. The networks we use in our current implementation are standard Ethernet and an FDDI ring.

Figure 1 depicts the architecture of the XMovie system. The main components of XMovie are the CM Server, the CM Client, the CM Agent and the Playback Application. These components can run on one, two or three different UNIX systems depending on the requirements of the application.

The **CM Server** is able to store and replay sequences of digital images (digital films). It maintains a movie directory. On request of the CM Client a sequence of images is sent over the network. The transmission protocol is called MTP (Movie Transmission Protocol) and was newly developed for the XMovie system.

The **CM Client** reads the movie stream from the network, performs forward error correction (FEC) and optionally decompression in software, and writes single frames into shared memory. All these frames are of constant size.

The **CM Agent** resides within the X server as an extension to the X code and reads frames at a specified rate out of shared memory. CM Client and CM Agent communicate through an extended X protocol.

The last component of the system is the **Playback Application**. It uses MCP (Movie Control Protocol), the control part of MTP, to communicate with the CM Client to control

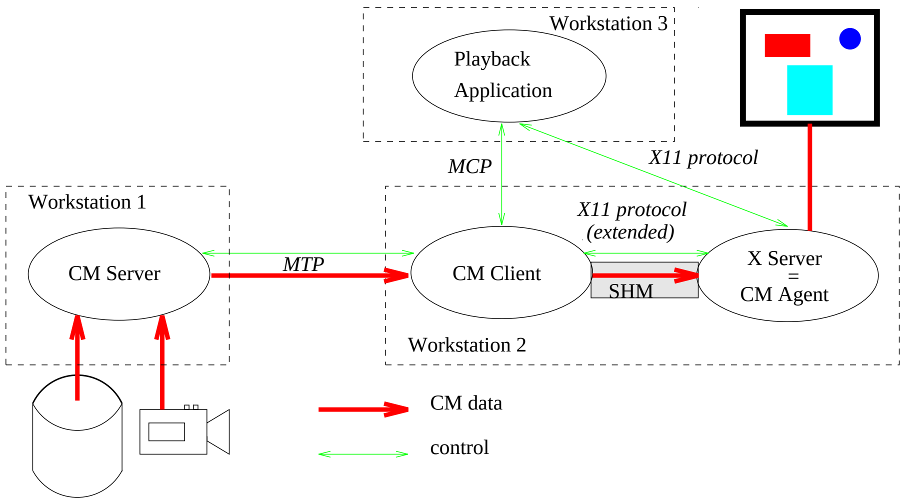

Figure 1: Architecture of the XMovie system

the playback of movies.

The main differences between the Berkeley Continuous Media Player [51] and XMovie are that CM Client and CM Agent run independently of each other during the playback of a movie, and that we use software decompression. The CM Agent receives only a start order and performs the reading and displaying of single frames. The CM Client writes frames to shared memory and does not have to inform the X server to display each frame each time. The queue introduced between CM Client and CM Agent is used for decoupling network and display and is the perfect place for jitter control. If both CM Client and CM Agent have enough CPU power available to fulfill their duty, i.e. to deliver and to display the frames at the requested rate, the system works very smoothly.

In this paper we concentrate on the *transmission* and *presentation* of digital movies; we do not discuss the creation or mass storage of such movies here. An introduction to the creation of movies can be found in [5, 30]. Digitized video films can be used as well as computer-generated digital movies. Storage aspects are discussed in [17, 49, 50, 61] and many other papers.

3 Formats, Compression Techniques and Display Methods for Movies

A number of image formats, compression techniques and display methods have been proposed for digital movies. As part of the XMovie design, we had to address these issues.

3.1 Formats for Digital Images and Movies

An image digitized with “true colors” uses 24 bits of storage per pixel (one byte for each of the color components red, green and blue). More than 16 million different colors can thus occur in one image. However, experience shows that the human eye cannot recognize small color differences [20, 38]. At the same time, “true color” representation needs too much storage to be practical (720 kbytes for a 600×400 pixel picture), so image compression is an obvious solution.

A very widely used form of image compression is the color lookup table technology. The graphics adapters of PCs and workstations can typically display 256 different colors at a time. The components for each color are stored in a color lookup table (CLUT or “color map”) containing 256 entries. Each pixel of an image is represented by one byte which is an index into the color lookup table (see Figure 2).

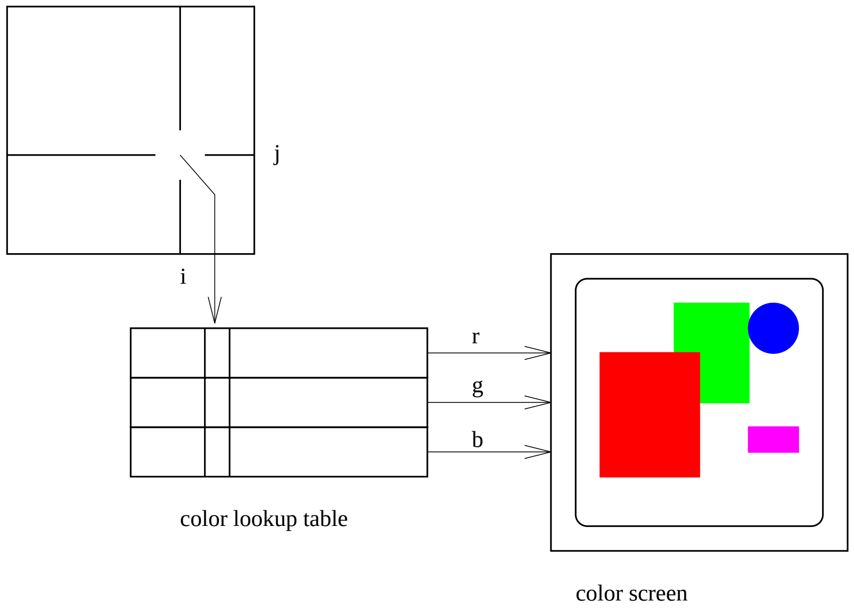

Figure 2: Color lookup table technology

For a specific image the colors in the CLUT can be optimized by choosing the most appropriate 256 colors out of a set of 16 million. Thus though only 256 colors can be presented at one time, selection is from a large set of colors. The corresponding CLUT is stored with each image. It reduces the amount of data to one-third without much loss of quality because the selection of colors is optimized for the color contents of the image.

3.2 Compression Techniques for Digital Images and Movies

Even with color lookup tables images are still very large, and further compression is desirable. We distinguish between lossless and lossy compression. *Lossless* compression implies that the decompressed image is identical to the original image. *Lossy* compression allows small differences, and usually provides much higher compression factors.

Compression algorithms can also be classified according to their time behavior. *Symmetric* algorithms use the same amount of time for compression and decompression, whereas *asymmetric* algorithms use more time for compression than for decompression. Asymmetric algorithms are especially useful in applications where compression can be done off-line, and the compressed images are stored for later use. Decompression is typically done much more frequently, and can often be done in real-time.

Image representation with a CLUT can be considered as a simple, lossy compression. Decompression is done in real-time in the graphics adapter. More sophisticated compression needs more computing power for decompression. This computing power can be provided with the following hardware approaches:

- a faster CPU
- special hardware for image decompression in the workstation or in the color graphics adapter
- multiple parallel processors.

An interesting compression scheme for still images is the Color Cell Compression algorithm (CCC, [7]; an extension can be found in [46]). CCC compresses to 2 bits/pixel. Because of the blockwise nature of the algorithm, compression and decompression can be parallelized well. The algorithm is lossy and symmetric. Our extension (XCCC) is described in detail in Chapter 3.4.

If we include the time dimension, movie formats can be further subdivided into two major groups: *intraframe formats*, and *interframe formats*. In pure intraframe formats a video is encoded as a sequence of independent frames. The encoding can either be uncompressed n-bit color images (n equals 1, 8 or 32), or compressed images such as XCCC, or JPEG [25]. The last case is known as Motion JPEG. Pure intraframe formats are robust to transmission errors because a single error is not carried over from one frame to another.

In interframe formats the compressed video stream consists of a combination of intra-coded and intercoded frames. Examples are MPEG [26] and H.261 [10]. For example, an MPEG compressed stream consists of a sequence of intra coded, predictive coded, and bidirectionally-predictive coded pictures.

All these algorithms are lossy, and JPEG and H.261 are symmetric, while MPEG is asymmetric. The compression factors are very high due to the Discrete Cosine Transformation and quantization steps, but compression and decompression require considerable computing power. Therefore the current trend is to build special compression/decompression chips. JPEG chips and MPEG chip sets are already available on the market, e.g. from C-Cube Microsystems [58].

The XMovie system has built-in decompression software for MPEG and H.261, in addition to the CLUT adaptation and XCCC schemes developed in Mannheim. The decoders are derived from the UC Berkeley MPEG player [45] and the INRIA H.261 videoconferencing system (IVS [57]). The support of the implementors is gratefully acknowledged. As mentioned above, for maximal flexibility and portability no decompression hardware is used in XMovie.

3.3 The DeltaCLUT Algorithm

Our first "home-made" movie format uses 8-bit color-mapped images, optimized for color lookup table technology in low-end workstations. There are three possibilities to generate a movie with CLUT out of a movie with full color:

- compute one CLUT for the whole movie

- compute a new CLUT for each frame
- compute a CLUT for the first frame and update it according to the color contents of subsequent frames.

The first two methods both have disadvantages. Using only one CLUT for the whole movie yields very poor colors. For a single picture 256 optimally chosen colors are usually sufficient, but not for a movie with many scenes, each with very different color contents. In the second method the workstation has to re-load the CLUT between two frames. The old frame is thus shown for a short time with the CLUT of the new frame. The effect is visually very disturbing. Loading the CLUT once the frame has been loaded results in the same effect: the new frame is shown with the CLUT of the old.

When the third method is used in a straightforward way, the same but somewhat weaker effect will be observed. XMovie extended the third method to avoid this problem: Each CLUT contains a small amount of free entries, i.e. no pixel of the frame uses these table entries (see Figure 3). The next frame can use and change these reserved entries. The actual frame can now be shown with its own CLUT and with the CLUT of the next frame without a visible difference, thus avoiding false colors.

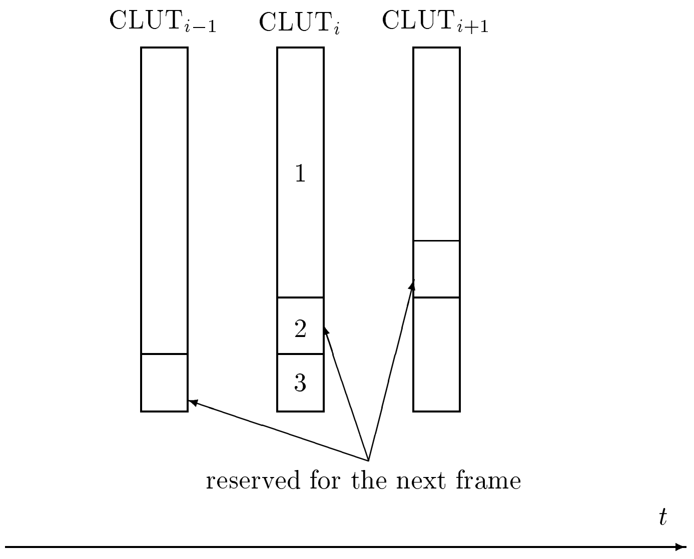

Figure 3: Free entries in the CLUTs

3.3.1 Computing Optimal CLUT Updates for a Given Movie

We have developed an algorithm called DeltaCLUT for the gradual adaption of the Color Lookup Table [33]. The CLUT for each frame i in the movie is computed as follows. Conceptually each CLUT has three classes of entries: Class 1 comprises the entries for colors identical in frames i-1 and i, class 2 comprises entries needed for frame i-1 but not occurring in frame i, and class 3 comprises new colors of frame i. The algorithm has four steps:

Algorithm DeltaCLUT

1. Choose a full CLUT for frame i with a conventional CLUT algorithm, for example median-cut.

2. Compare the colors of CLUT i with the colors of CLUT $i-1$. Identical or nearly identical colors are re-used from CLUT $i-1$ (class 1).

3. Copy all CLUT entries used in frame $i-1$ but not yet in CLUT i to CLUT i (class 2). These colors are not needed by frame i, so that frame $i+1$ can overwrite them later.

4. Choose some more colors for CLUT i and load them into the entries of class 3.

In the first step the standard technique median-cut is used to map the full-color image onto a CLUT image [22]. The second step of the algorithm searches for nearly identical colors in CLUT i-1 according to parameter one. The colors found are copied to CLUT i (class 1). In step three all other colors used in frame i-1 are copied to CLUT i, hence all colors used in frame i-1 are now in CLUT i, but the colors copied in step three are marked as not usable (class 3). Later they can be overridden by CLUT i+1 without disturbing frame i. In the fourth step more colors remaining from the output of the median-cut algorithm are added until CLUT i is filled.

The first frame is a special case. It is handled by reserving a fixed but arbitrary number of entries and filling all others with colors of frame 1 from the median-cut algorithm.

Extensive measurements were performed in order to evaluate the influence of various parameters on the visual quality of the movie. The details are described in [34]. Our experiments showed that the color quality of single raytraced frames is only slightly poorer compared with pictures using all 256 colors.

The DeltaCLUT algorithm is rather time consuming due to the embedded median-cut algorithm. The DeltaCLUT part itself is quite fast. The time is nearly independent of the size of the frames because the amount of different colors in a frame is independent of its size.

The dithering of the frames can be done with any standard dithering algorithm. The time is linear with the number of the pixels in the movie because the algorithm has to find the index for each pixel.

3.4 eXtended Color Cell Compression (XCCC)

The standardized compression techniques JPEG for still images and MPEG for motion pictures both include a Discrete Cosine Transform (DCT). This is a complex and computationally very demanding mathematical function. As a consequence, software motion pictures based on JPEG or MPEG are slow, even on the most powerful CPUs available today, and it is generally assumed that these compression schemes will only work well with special hardware. However, special hardware makes a movie system much less flexible and portable. It is thus reasonable to also consider alternative compression/decompression algorithms for movies which are optimized for computation in software on general purpose CPUs.

We propose an extension to the Color Cell Compression scheme, called XCCC, for use in multimedia workstations. After a short introduction into the predecessors of XCCC, Block Truncation Coding for monochrome images and Color Cell Compression for color images, we describe XCCC in detail.

3.4.1 Block Truncation Coding (BTC)

The Block Truncation Coding Algorithm [15] is used for the compression of monochrome images. The first step of the algorithm is the decomposition of the whole image into blocks of size $n \times m$ pixels. Usually these blocks are quadratic with $n = m = 4$. For each block P the mean value μ and the standard deviation σ is computed:

$$\begin{aligned} \mu &= \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} P_{i,j} \\ \sigma &= \sqrt{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} P_{i,j}^2 - \mu^2} \end{aligned}$$

where $P_{i,j}$ is the brightness of the pixel.

In addition a bit array of size $n \times m$ is calculated for each block. A 1 in this bit array indicates that the gray value of the corresponding pixel is greater than the mean value, a 0 indicates that the value is less than the mean value:

$$B_{i,j} = \begin{cases} 1 & \text{if } P_{i,j} \geq \mu \\ 0 & \text{else} \end{cases}$$

The decompression algorithm knows out of the bit array whether the pixel is darker or brighter than the average. Last we need the two gray scale values for the darker and for the brighter pixels. These values (a und b) are calculated with the help of the mean value and the standard deviation, and are then stored together with the bit array:

$$\begin{aligned} a &= \mu + \sigma\sqrt{p/q} \\ b &= \mu - \sigma\sqrt{q/p} \end{aligned}$$

Here p and q are the number of the pixels having a larger resp. smaller brightness than the mean value of the block.

During the decompression phase each block of pixels is calculated as follows:

$$P'_{i,j} = \begin{cases} a & \text{if } B_{i,j} = 1 \\ b & \text{else} \end{cases}$$

e.g. where the bit array shows a 1, the gray value a is used, where it shows a 0 the value b is used.

If the original image used one byte per pixel, we had a storage requirement of 128 bits for each 4×4 block. The compressed block can be stored with 16 bits for the bit array plus one byte for each of the values a und b. Hence we have a storage reduction from eight bits to two bits per pixel.

This basic version of BTC can be improved with a number of tricks [46]. Additionally, [52] describes a hierarchical version of BTC.

3.4.2 Color Cell Compression (CCC)

If BTC is to be used for color images rather than for gray scales, the components (red, green, and blue, resp. chrominance and luminance) may be compressed separately. However, the CCC method promises a much better compression rate [7].

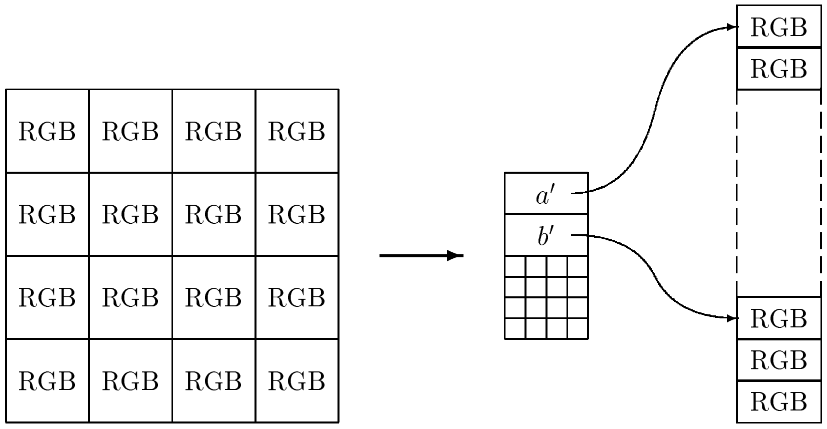

Figure 4: Red-Green-Blue block and its CCC encoding

Similarly to BTC, the image is divided into blocks called "color cells". The two values a and b are now indices into a color lookup table (CLUT). The criterion for the bit array values is now the brightness of the corresponding pixels. The brightness of a pixel is computed in the following way, taking human perception into account:

$$Y = 0.3P_{\text{red}} + 0.59P_{\text{green}} + 0.11P_{\text{blue}}$$

The mean value of each block can now be computed out of these brightness values (analogous to the BTC method).

Let us define $P_{\text{red},i,j}$ as the red component of $P_{i,j}$, $P_{\text{green},i,j}$ the green component and $P_{\text{blue},i,j}$ the blue component. The next step is then to compute the color values a_{red}, a_{green}, a_{blue} as well as b_{red}, b_{green}, b_{blue}:

$$a_c = \frac{1}{q}\sum_{Y_{i,j}\geq\mu} P_{c,i,j} \;\text{ bzw. }\; b_c = \frac{1}{p}\sum_{Y_{i,j}<\mu} P_{c,i,j} \;\text{ with } c = \text{red, green, blue}$$

Again p and q are the number of pixels with a brightness larger resp. smaller than the mean value. The bit array is computed as for BTC.

The color values $a = (a_{\text{red}}, a_{\text{green}}, a_{\text{blue}})$ und $b = (b_{\text{red}}, b_{\text{green}}, b_{\text{blue}})$ are now quantized onto a color lookup table. In this way we get the values a' und b'. These values are stored together with the bit array (Figure 4).

The decompression algorithm works analogous to the BTC method:

$$P'_{i,j} = \begin{cases} \text{CLUT}[a'] & \text{if } B_{i,j} = 1 \\ \text{CLUT}[b'] & \text{else} \end{cases}$$

(CLUT is the color lookup table.)

The two values a' und b' can each be stored in one byte if the CLUT has 256 entries. Hence the storage needed for one block of size 4×4 is two bits per pixel as with the BTC (to be more exact we would have to add the storage needed by the CLUT (256×3 bytes for the full image)).

Color Cell Compression is not only one of the best compression algorithms, it is also one of the fastest [46]. All calculations can be done without floating point operations, and the asymptotic complexity is $\mathrm{O}(N \cdot M \cdot (1 + \frac{\log k}{n \cdot m}))$ (image size $N \times M$, size of the block $n \times m$, size of the CLUT k). The decompression is also done without floating point operations with a complexity of $\mathrm{O}(N \cdot M)$.

As in the case of the BTC algorithm, a number of possible improvements exist here as well [46]:

- If the two colors a and b are nearly equal, or one color dominates in frequency of occurrence, only one color is stored, and no bit array is needed.
- If an image contains large areas with only small differences in color, those areas may be encoded with larger blocks.
- For movies cuboids may be used, with time being the third dimension if the changes from frame to frame are small enough.

3.4.3 XCCC: Extensions to CCC

Our XCCC scheme extends CCC in three steps in order to improve compression ratio and runtime performance.

First step: Adaptive block sizes. In many cases an image has large areas with small differences in colors (i.e. in the background). These areas can be coded with fewer bits.

We first investigate the optional use of large rectangles. If an image contains large areas with few colors, these areas can be compressed with larger rectangles.

In XCCC the images are first decomposed into large blocks (16×16) and, if necessary, these blocks are then subdivided. The algorithm for each block B is:

1. Calculate the CCC coding of the block
2. If the actual block has the minimal block size, then Done.
3. Calculate the mean difference Δe of the original pixel values and the values coded with CCC: $\Delta e = \sum_B |p - p'|_2$, where p is the pixel value, p' is the value of the same pixel after decompression.
4. If Δe is smaller than a given constant, then Done.
5. Divide the block into four subblocks and use the algorithm recursively for each of these blocks.

The data for simple CCC could be arranged in the data stream without any structuring information. But the output of the extended algorithm is a quadtree with color cells for each 16×16-block. Hence, we have to store a more complex data structure. This is done by adding a tag for each block. Figure 5 shows an example of the coding of an XCCC block. The tag is the logarithm of the length of the edge of the coded block. Blocks of minimal size need only one tag for four blocks because one minimal block is always followed by three more. After the tag we store the indices into the CLUT (a and b) and then the bit array. If the length or width of the image is not divisible by 16, we divide the residual rectangle into 4×4 blocks and encode some of these subblocks as rectangles.
The use of adaptive block sizes introduces a small additional overhead for compression and results in much more efficient decompression for most images.

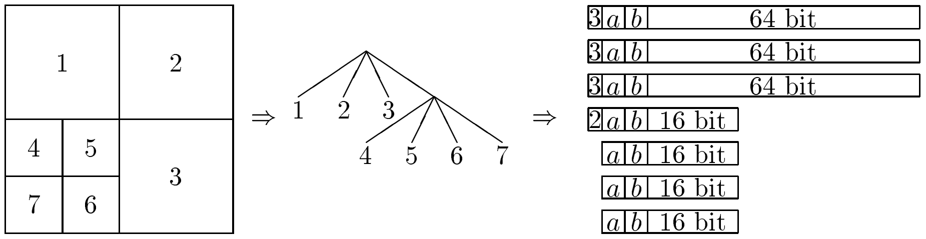

Figure 5: Coding tree of the XCCC-algorithm

Second step: Single Color and Color Reuse. If an image has a large area with only one color, it is not necessary to store the bit array at all. XCCC does not store a bit array if the two color indices a and b are equal. There are two ways to let the decoder know that there is no bit array: First the encoder could store the two colors and the decoder would know from the equality that there is no bit array. Second, the encoder could use a special tag and store only one color. XCCC uses the second method.

If we implement bit array suppression for single color blocks, larger squares will not always improve the compression ratio. Instead of a 32×32 square, XCCC may use four 16×16 squares. But some of these squares will have no bit array and hence the total compression ratio may be better. Although this can also happen with smaller blocks, our experience shows that this is rarely the case.

Colors in the spatial neighborhood are often equal in images. Because of this fact, we can sometimes reuse colors from the block coded previously. Color reuse is also stored in the tag.

Third step: Further improvements. Some blocks are encoded in only one or two bytes, namely the blocks without bit array. If one of the four subblocks of a 16×16 block is a single color block, then it is cheaper in memory to store the four subblocks instead of the 16×16 block. Some are even stored in one byte only, namely those where a or b are taken from the previous block.

The remaining redundancy in the bit stream could be further compressed with Huffman codes [24]:

- The bit stream consist of three parts: The tags, the colors and the bit arrays. All three parts still have redundancy in the stream.
- Some of the tags are used very often, others never or very seldom. The usage of a Huffman code will reduce the total size of the tags to 50%.
- The colors can be compressed only by about 10% with a Huffman code because of the usage of a color lookup table. This table is chosen to allow use of all colors equally often. Hence only a small redundancy of about 10% remains.
- A large redundancy is in the bit arrays, especially in the bit arrays of the 4×4-blocks. The redundancy can be as much as 60%.

Hence compression of the tagged stream with three different Huffman codes would divide the size into half, but also slow down decompression.

From frame to frame the color of a pixel seldom changes. So we could use a second "same color" flag to signal the reuse of the corresponding color from the previous image.

A third improvement has been implemented: All blocks are test-wise subdivided into 4×4 blocks. Step 4 of the XCCC algorithms is then changed to the following:

4. If Δe is smaller than a given constant, and encoding of the actual block is better than encoding of the subdivided block, then Done.

3.4.4 Experience: Compression Ratio and Decompression Speed

The main goal of the XCCC scheme is to allow rapid decompression by software decoders. Table 1 shows the decompression speed in images per second for XCCC and MPEG. The MPEG measurements were done with the MPEG player of the University of California at Berkeley. We used three movies: The first and second movie (butterfly) are raytraced sequences of 350 frames, with size 320×256 and 780×576 resp. The third movie was digitized from an analog video showing the University of Mannheim. It consists of scenes of buildings of the university (an old palace) and other scenes depicting university life. Due to the analog origin, this movie consists of many different colors and color shades. It is 320×256 in size and has 2000 frames.

The decompression was done on a DEC AXP workstation with a 133Mhz CPU. The speeds are the real speeds viewed on the screen. The display adapter uses a color lookup table with 8 bits per pixel. XCCC uses the same technique internally thus requiring no conversion. In contrast, MPEG uses full color internally. Hence, the MPEG player has to dither in real-time. The player has several built-in dithering methods. For the tests we used the fastest color dithering available ("ordered dithering").

Movie	Size	MPEG	XCCC
	(pixels^2)	(frames/s)	(frames/s)
Butterfly	320×240	10.5	42
Butterfly	780×576	2.1	6.8
Castle	320×240	7.8	24

Table 1: Decompression speed of software decoders (in frames/s)

Movie	Size	MPEG	JPEG	XCCC
Butterfly	320×240	0.80%$\hat{=}$0.19bpp	2.49%$\hat{=}$0.60bpp	3.0%$\hat{=}$0.72bpp
Butterfly	780×576	0.53%$\hat{=}$0.13bpp	1.54%$\hat{=}$0.37bpp	1.83%$\hat{=}$0.44bpp
Castle	320×240	1.5%$\hat{=}$0.36bpp	5.9%$\hat{=}$1.42bpp	6.3%$\hat{=}$1.51bpp

Table 2: Compression ratios (compressed size/full color size and bits per pixel)

The compression speed was measured on a DECstation 5000/133. For the small butterfly movie we got about 6.5 seconds per image with MPEG. Before XCCC can be started, the color lookup tables have to be computed with DeltaCLUT (see 3.3). This step takes about 7 seconds per image. XCCC then takes about 2 seconds per image. Together our compressor uses about 10 seconds per image.

Our experiments show that XCCC can decompress images very fast. The quality of the XCCC images is comparable to the quality of MPEG images. The great advantage of MPEG is the bit rate of the compressed movie: The size of XCCC compressed movies is about three to four times larger than the size of MPEG movies. Hence the domain of XCCC are environments with workstations using the color lookup table technique, where bandwidth and storage requirements are less critical than runtime performance and flexibility. In this environment XCCC performs significantly better than MPEG.

3.5 Integration of Movies into a Window System

For the workstation user the movie should appear in a window on the screen without disturbing other windows. The handling of the movie window should be similar to that of other windows. Real movie windows are not yet supported in current window systems, such as the X window system. An integration of movie windows can be done in the following ways:

- Hardware can be added (blue-box or real-time digitization of analog video sources), e.g. Pandora's box [23] or DVI [19]. As we have argued above, the special hardware needed limits flexibility and makes integration with other windows difficult; it also limits the number of open movie windows.
- The window software is able to show single digital images. This property is abused to throw fast sequences of still images at it, and it will try to draw as fast as it can. Examples are the MPEG player of UC Berkeley [45], the INRIA H.261 videoconferencing system (IVS) [57], and nv [8]. The main disadvantages of this approach are that synchronization is up to the application, that the Playback Application and X server must be co-located on the same system to gain highest speed, and that multiple movies create considerable overhead.
- The window system is extended by a native movie window. This is the approach taken in XMovie.

In the case of the X window system this implies full control of the movie window by the X server process. The X server is able to process simultaneously several movie orders from CM Clients. For each order it reads a stream of frames at a specified rate out of a frame queue in shared memory (see Figure 1). All movie playbacks have to share available system resources, in particular SHM segments.[1] But this is not a severe restriction since we do not believe that more than one or two movie playbacks will happen at a time, and in live video only a small number of frames can be buffered since live video requires low end-to-end delay. More details about our extension to the X window system can be found in [28].

We see many advantages compared to the second solution above:

- The communication overhead between X client, resp. CM Client and X server is lower.

[1]The number of SHM segments attachable by a process is limited by the operating system.

- The code for movie presentation (i.e. presentation of sequences of images) does not have to be repeated in each Playback Application.
- The X server can be better adapted to the workstation hardware, so that performance and hence the movie quality is better.
- The manipulation of the movie window (open, close, zoom, ...) works as usual (for the user and for the programmer).
- Jitter, added to the continuous media stream by transmission, decoding, or other functions, can be removed easily in the shared memory queue.
- The CM Client can perform flow-control by observing the fill-level of the shared memory queue.

4 Application Layer Issues

Multimedia applications can be written more easily if powerful services are available within the application layer. Besides continuous media streams (CM streams) we have addressed two such services: movie directory and CM equipment control. The movie directory is used as a repository for movie information, such as digital format and storage location. The equipment control service enables the user to control CM equipement attached to local or remote computer systems, e.g. speakers, cameras, and microphones. We have defined MCAM (Movie Control, Access, and Management), an application layer protocol which provides these three services to the user. Since we focus in this paper on the transmission and presentation of digital movies, only the CM streams part of MCAM is presented here. More details about MCAM can be found in [27, 29].

4.1 The Movie Transmission Protocol MTP

The CM stream service allows isochronous transmission of CM streams over high-speed networks. For this service we have developed MTP. To provide this isochronous transfer service in a heterogeneous environment MTP has to extend the functionality of existing transport interfaces by forward error correction and rate based flow control, and has to keep delay and delay jitter low. In addition, MTP should be portable and extendable to allow an easy adaptation to new requirements and new transfer services.

The place of MTP in the OSI reference model is shown in Figure 6. MTP is an application layer protocol with integrated presentation functionality, and is built upon the Internet protocols TCP [48] and UDP [47].

Other protocols for the transmission of CM streams are defined in the literature, e.g. [6, 14, 54, 60]. All these are transport layer protocols. In contrast, the architecture of XMovie is based on common transport services and moves application-specific functions into the MTP layer. This improves the portability of the system. It would be desirable to have available a transport subsystem for continuous streams. Many such systems are under development, but at this time we are unhappy with all of them. They either lack resource reservation and thus guaranteed quality of service, or they have no multicast, or no dynamic join and leave for receivers. We are eagerly awaiting the next generation of CM transport protocols.

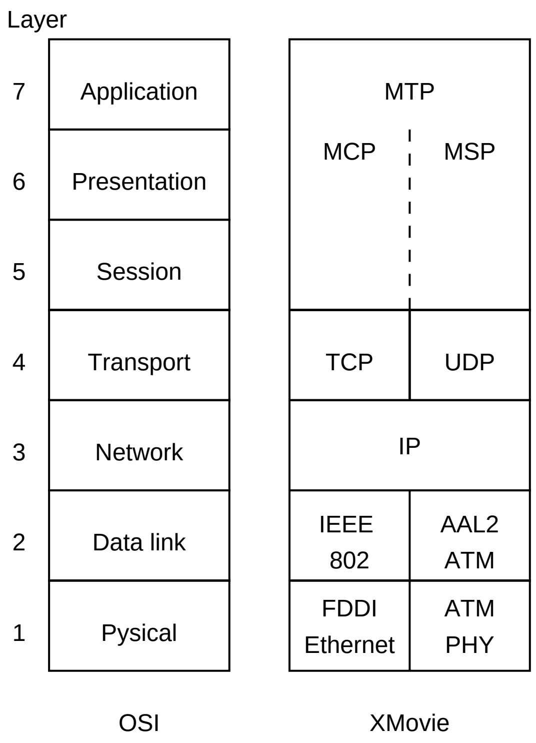

Figure 6: Classification of MTP in the OSI reference model

MTP can be used for control purposes only or for combined control and transmission of CM streams. In Figure 1 only control messages are exchanged between the Playback Application as Initiator of and the CM Client as Responder to a connection. However, between the CM Client as Initiator and the CM Server as Responder, control messages as well as CM data are transmitted.

MTP can be subdivided into two parts: the control protocol MCP (*Movie Control Protocol*) and the stream protocol MSP (*Movie Stream Protocol*). There are several reasons for such a subdivision: Control protocols have different requirements for underlying services than do CM stream protocols. A control protocol needs a reliable asynchronous transport service with low data rate, and has only weak requirements on the delay jitter. In contrast, a CM stream protocol requires high bandwidth and isochronous transmission with low delay. Data loss can often be tolerated. The acceptable loss rate is application-specific and should be adjustable.

Also, the separation into two parts allows the use of MTP in event-driven and in media-driven applications. Both kinds of applications use different control structures to provide the requested service. Event-driven applications, supporting user interfaces and higher program functions, have to react to sudden events immediately, no real-time requirements are given. In contrast, media-driven applications, where the main part of processing takes place, needs to be performance-optimized and real-time quarantees have to be fulfilled. An example of an event driven application is the Playback Application in Figure 1, using the MCP service only. Both CM Client and CM Server are examples of media-driven applications; they use the MCP and MSP services.

With TCP the Internet protocol suite offers a reliable connection-oriented transport protocol. MCP uses TCP for the transmission of its PDUs. TCP cannot be used for isochronous transmission because its error correction and flow control algorithms add large delay and delay jitter [32]. Therefore MSP uses the unreliable connectionless transfer service of UDP for the transmission of CM streams.

MSP needs to extend the service of UDP to provide a CM stream service. It adds forward error correction and rate-based flow control. Since UDP does not provide an isochronous transfer service, jitter added to the CM streams by the network needs to be eliminated. In XMovie we decided to remove jitter as close to the final sink as possible, since decompression and other functions can add jitter to the stream immediately before the display, e.g. in MPEG the decompression of an I frame takes much longer than the decompression of a B frame. As mentioned in Chapter 3.5, the shared memory queue of frames between CM Client and CM Agent is used to eliminate jitter.

Jitter is added to the CM stream by the operating system and the network protocol. In order to solve the operating system problem, the OS scheduler must be modified to take the requirements of real-time processing into account. More on this topic can be found in [1, 21, 41, 43].

As far as the network protocols are concerned, the problems are much more difficult to solve because almost all layers introduce variable delays. It starts out with media access protocols in LANs. For example, in a CSMA/CD-based protocol, carrier sensing can lead to variable delays, depending on traffic from other stations on the bus. Also, collisions can delay packet transmission in an unpredictable way. Similarly, a Token Ring LAN will have variable packet delays depending on where the token happens to be when transmission is requested, and on how many priority stations are waiting for transmission. The same is true for the token-based media access control protocol of FDDI.

Conventional transport protocols are not suitable for isochronous transmission either. Both TCP and ISO TP4 use time-outs and retransmission for error correction. Of course retransmitted packets will have a longer delay than first-shot packets. Fortunately the new high-speed networks now under development, such as ATM, will provide protocols for isochronous data flow [12]. At the same time, work is in progress to extend the Internet protocol suite to enable isochronous data flows [9, 13, 18, 54, 59, 62].

4.2 Forward Error Correction

Apart from the XCCC compression scheme discussed above, *adaptive forward error correction* is another novel feature of XMovie. The error handling method in traditional communication protocols is error detection and retransmission. This is inappropriate for distributed multimedia systems for two reasons: It introduces variable delay unacceptable for isochronous streams, and it is very inefficient and difficult to use in the multicast environment typical for many multimedia applications. We propose AdFEC, an adaptable Forward Error Correction scheme based on binary polynomial algebra. It produces an adaptable amount of redundancy, allowing different packet types to be protected according to their importance.

Single bit errors rarely occur in modern networks, especially if they are based on fiber optics. The main source of errors is packet loss in the switches. Current procedures that focus on the correction of individual bit errors do not solve this problem.

Very few articles address the problem of reconstructing lost packets [4, 40, 44, 55]. These articles deal with packet loss in ATM networks. All packets in the data stream are protected by means of the same method, and with the same redundancy.

The AdFEC scheme developed for XMovie is capable of assigning different priorities to different parts of the data stream. The amount of redundancy for FEC is chosen according to the priority. A digital data stream for a movie or for audio contains more than just the digitized image/audio contents. It also contains information that must not be lost under any circumstances, such as control instructions, format data, or changes in the color lookup table. Typically a higher error rate can be tolerated for content parts than for control parts, but all packets have to arrive on time. For example, we can assign priorities as follows:

Priority 1: Segments that may not be lost under any circumstances (e.g. control and format information as well changes in the color lookup table)

Priority 2: Segments whose loss clearly adversely affects quality (e.g. audio)

Priority 3: Segments whose loss is slightly damaging (e.g. pixel data in a video data stream).

For none of the three priorities is retransmission a tenable option. Starting from an already low rate of loss, third-priority packets can be transmitted without protection, second-priority packets should be protected by means of FEC with minimum redundancy, and first-priority packets by means of FEC with high redundancy.

4.2.1 Creating Redundancy for Forward Error Correction

Traditional error-correcting codes (e. g. Reed-Solomon) were designed for the detection and correction of bit errors [36]. Since the demand now exists to also restore entire packets, new codes must be found. Specifically, errors need no longer be located, the lost bits are known. A feature of traditional error-correcting codes is their ability to locate damaged bits. The price of this feature is a great deal of redundancy. At issue here is to devise a code that restores the lost packets at a known error location.

Example 1: Two packets p_1 and p_2 are to be sent. A redundancy of 100% is taken into account, i. e. two additional packets may be generated. These additional packets are sent together with the original packets. In the event of packet loss, the original packets p_1 and p_2 must be restored from the remaining packets (see Figure 7). In this case two operations (labeled $\circ$ and $\bullet$) are necessary, with whose help the redundant packets can be generated.

We can now define the problems in mathematical terms.

Definition: Bit sequence $\mathbb{B}_l = \{0, 1\}^l$, for fixed $l \in \mathbb{N}$.

given: n packets $P = \{p_1, p_2, \ldots, p_n \in \mathbb{B}_l\}$

find: $n + m$ packets $Q = \{q_1, q_2, \ldots, q_{n+m} \in \mathbb{B}_l\}$ such that upon the arrival of at least n packets out of Q all packets of the set P can be restored.

Example 2: $n = 2, m = 1$: Choose $q_1 = p_1, q_2 = p_2, q_3 = p_1 \oplus p_2$. $\oplus$ is the operator for binary exclusive or (XOR). If, for example, q_1 is lost, p_1 can be restored with $p_1 = q_2 \oplus q_3$.

A total of 16 binary operators can be defined combining two bits, namely all possible combinations of zeroes and ones in a four-row value table. In order to construct our packets

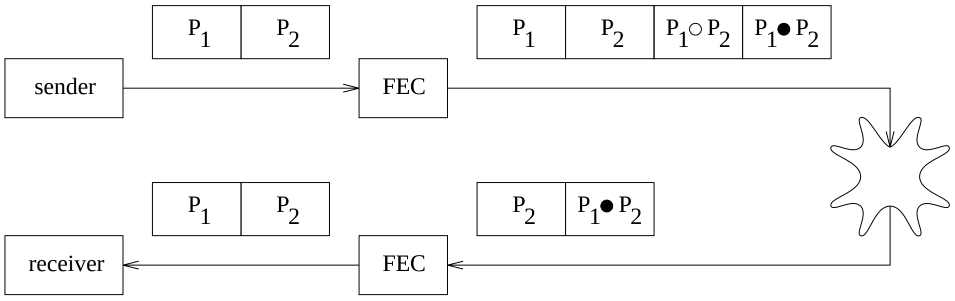

Figure 7: Principle of forward error correction for packet losses (assume the network loses the packets p_1 and $p_1 \circ p_2$)

for Q, operators are required whose result can be used for the unambiguous reconstruction of any bit fields. The only binary operator suitable for this is the XOR-operator and its negation, the equivalence. Therefore on the basis of just the 1-bit operators only one redundancy packet can be generated, all other packets would not allow an unambiguous reconstruction in all cases.

In order to generate additional independent packets, a field containing 2^n elements $(n > 1)$ must be sought. Such fields can be generated with the aid of polynomial algebra [36]. Table 3 shows the operations $+$ and $*$ for a field containing four elements.

$+$	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

$*$	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

Table 3: The operations $+$ and $*$ for a field containing four elements

For $n = m = 2$ the operators $\circ$ and $\bullet$ can, for example, be defined as follows (the field is labeled GF for Galois Field):

$$a, b \in \mathrm{GF}(2^2) \Rightarrow \quad \begin{aligned} a \circ b &= a + b \\ a \bullet b &= a + 10 * b \end{aligned}$$

The two redundancy packets are thus defined as

$$c = a + b$$

and

$$d = a + 10 * b$$

In the case of loss of two data packets, the calculations in Table 4 are to be carried out.

arriving	Calculations for reconstructing	
Packets	a	b
a, b	-	-
a, c	-	$a + c$
a, d	-	$11 * (a + d)$
b, c	$b + c$	-
b, d	$d + 10 * b$	-
c, d	$11 * c + 10 * d$	$10 * (c + d)$

Table 4: Calculations for reconstruction of lost packets

Returning to our notation above we set $p_1 = a, p_2 = b, q_1 = c$ and $q_2 = d$. Thus, if any two packets are lost, the contents of p_1 and p_2 can be reconstructed from the arriving packets.

Generally speaking, fields containing p^n elements (p is prime, $n > 0$) can be derived. Of particular interest to the computer scientist are of course those with $p = 2$. For this reason only such fields will be considered in the following. The book by Lin and Costello [36] provides in its appendix a list of the possible generator polynomials for GF(2^n) to $n = 10$.

4.2.2 The Power of Polynomial Algebra for Forward Error Correction

The fields constructed in the preceding section are to be examined regarding their power for generating redundant packets. Of particular interest is the number of *independent* packets that can be generated. Given the two packets a and b, generation occurs in the form $\lambda a + \mu b$, with $\lambda, \mu, a, b \in$ GF(2^n).

Thus, given a field GF(2^n) and a number of packets $a_1, \ldots, a_k$, the question is: How many independent packets l can be generated by the given field from k packets? The more packets generated and transmitted, the greater the probability that the recipient will be able to reconstruct the packets sent from those that arrived. It has already been explained above that by means of a field GF(2) only one single additional packet can be generated, independent of the number of given packets. The extension to the field GF(2) apparently is of little use in this respect. For this reason the fields GF(2^n) are examined.

Generally, the contents of the output packets for two input packets a and b adhere to the following scheme:

Output Packets:

$p_{ij} = \lambda * a + \mu * b$ with $i, j = 1 \ldots 2^n$ and $\lambda, \mu \in$ GF(2^n) (see Table 5)

Some of the packets generated in this manner are linearly dependent, and their transmission thus affords no advantage for reconstruction. The transmission of the first packet is obviously useless because it is always equal to zero. The pairs of linearly dependent packets can now be combined into classes. $n-1$ elements belong to every class, in the example three elements of the table. The result yielded here is the set $\{\{b, \alpha b, (1+\alpha)b\}$, $\{a+b, \alpha a+\alpha b$, $(1+\alpha)a+(1+\alpha)b\}$, $\{a+\alpha b$, $\alpha a+(1+\alpha)b$, $(1+\alpha)a+b\}$, $\{a+(1+\alpha)b$, $\alpha a+b$, $(1+\alpha)a+\alpha)b\}$, $\{a, \alpha a$, $(1+\alpha)a\}\}$.

A more detailed discussion of the systematic construction of linearly independent packets, with several theorems and algorithms, can be found in [31].

Let us now come to the corrective power and the overhead of our scheme. The more

1.	$0 + 0$	9.	$\alpha a + 0$
2.	$0 + b$	10.	$\alpha a +$ b
3.	$0 + \alpha b$	11.	$\alpha a + \alpha b$
4.	$0 + (1+\alpha)b$	12.	$\alpha a + (1+\alpha)b$
5.	$a + 0$	13.	$(1+\alpha)a + 0$
6.	$a +$ b	14.	$(1+\alpha)a +$ b
7.	$a + \alpha b$	15.	$(1+\alpha)a + \alpha b$
8.	$a + (1+\alpha)b$	16.	$(1+\alpha)a + (1+\alpha)b$

Table 5: Generateable packets in GF(2^2)

linearly independent packets we include in the transmission, the higher the corrective power (i. e. chance of restoration), but the higher, also, the overhead. For example, only minimal protection is provided by a single additional packet for every ten data packets. This results in a redundancy of only 10 percent. But with our scheme it is now easy to generate a larger number of redundant packets, thereby strongly increasing the reconstruction probability. For example, using a code out of the field GF(2^8), 257 pairs of linearly independent packets could be generated out of only two data packets. This goes to the other extreme in increasing the data rate, but at the same time provides much better correction probability.

As we have seen, the method described enables the generation of an *adaptable redundancy*. Using the same method of calculation at the sender and receiver, the different segments of the data stream can be protected with varying degrees of correction probability and overhead. Therefore, we call our method AdFEC (Adaptable Forward Error Correction).

4.2.3 The Corrective Power of AdFEC

In this section we analyze the corrective power of the AdFEC method. We compare AdFEC with two alternatives: duplicating each packet, or no redundancy at all (i. e. no error correction).

Let n be the number of packets to be sent and m be the number of packets that are added for error correction. Furthermore, let q be the probability that a packet arrives (i. e. $1-q$ is the probability of loosing a particular packet). We assume that the loss of packets is an independent process. We use the following notation to denote the probabilities that the original n packets can be recovered:

$p(n,m)$	under the assumption that m additional packets are added;
$p_d(n)$	under the assumption that every packet is send twice;
$p_n(n)$	no forward error correction.

To obtain a formula for $p(n,m)$ note that the probability that exactly i out of the $n+m$ packets get lost is

$$q^{n+m-i}\,(1-q)^i \binom{n+m}{i}$$

. In order to reconstruct the original n packets, at most m packet losses are tolerable. Taking the sum over all possible cases we get

$$p(n,m) = \sum_{i=0}^{m} q^{n+m-i}\,(1-q)^i \binom{n+m}{i}$$

For the analysis of $p_d(n)$ note that the original n packets can be reconstructed only if at least one of the two duplicates arrives. Thus, if i packets are lost, the original n packets arrived at the receiver if the lost packets were all different. There are

$$2^i \binom{n}{i}$$

different cases for losing exactly i different packets. This yields

$$p_d(n) = \sum_{i=0}^{n} q^{2n-i} (1-q)^i \, 2^i \binom{n}{i}$$

Clearly for the function $p_n(n)$ we have

$$p_n(n) = q^n$$

The overhead in the case of sending every packet twice is 100%. In the case where m additional packets are sent the overhead is 100m/n%.

The following tables characterize the situation for $q = 0.9$ and $q = 0.7$. In the first case, we have $p_n(10) = 0.35$ and $p_d(10) = 0.90$, and in the second case we have $p_n(10) = 0.03$ and $p_d(10) = 0.39$.

Probability $q = 0.9$

numb. of add. packets	$p(10, m)$	overhead in %
1	0.697	10
2	0.889	20
3	0.966	30
4	0.991	40
5	0.998	50

Probability $q = 0.7$

numb. of add. packets	$p(10, m)$	overhead in %
1	0.113	10
2	0.253	20
3	0.421	30
4	0.584	40
5	0.722	50

Thus, in the case $q = 0.9$ with an overhead of only 20%, nearly the same arrival probability can be achieved by our algorithm as in the case where every packet is sent twice (i. e. the overhead is 100%). In the case $q = 0.7$ the situation is similar: with an overhead of only 30% a better result can be achieved than with duplication. Moreover, the arrival probability with only 10% overhead is 69% (11%), compared to 35% (3%) without error correction. Hence, our adaptable method is superior to the simple method of duplicating the packets, and with very little overhead it is possible to improve the arrival probability by a factor of 5 – 10 over no error correction.

4.2.4 Performance of AdFEC

We have implemented the AdFEC algorithms described above in the XMovie system. In the current implementation, we can generate for n given packets, $n \in \{1, 2, 3\}$, m redundant packets, $m \in \{1, 2\}$. It might be a good choice to protect single frames out of a Motion JPEG movie to guarantee a certain quality. In MPEG coded movies I frames could be protected be a high redundancy and P frames with a low redundancy. B frames could be transmitted without error protecting redundancy.

In the framework of XMovie's MTP protocol, AdFEC is used to protect parts or all of the CM stream. Because reconstruction requires only XOR and multiplication uses small tables in memory, AdFEC could be implemented with very few machine instructions. Table lookup is more efficient than explicit computation at runtime, and can be carried out in just a few machine instructions. Since the addition corresponds to the XOR operation which is carried out in hardware, the total efficiency of AdFEC is very high. AdFEC was written in C++ on different UNIX workstations. Because standard C++ was used exclusively, porting of the error correction procedure to other architectures is very easy.

5 Implementation and Experience

Since the beginning of the XMovie project, two major prototypes of the system were developed. Both are in the public domain and can be used by other research groups.[2]

5.1 Initial Implementation of XMovie

The first implementation of XMovie has been operational since August 1992. It was initially implemented on an IBM PS/2 Mod. 80 under AIX in standard C and was ported to DECstation 5000, Sun SPARCstation 10 and IBM RS/6000 [32]. The movies were transmitted over Ethernet or FDDI, using IP and UDP and a first version of MTP on top of these. The movies we used for experimentation were digitized videos and computer-generated films. The only movie format in the initial XMovie implementation was the DeltaCLUT format.

5.2 Extensions of XMovie

To remove performance bottlenecks in the initial implementation of XMovie and to integrate new features such as multiple movie formats, jitter elimination, rate-based flow control, and AdFEC, we have redesigned the system architecture as described in [28].

Many of the parts of XMovie are now realized as C++ classes, e.g. MTP, AdFEC, and all decoders.[3] Now the implementation can be maintained and extended more easily. For example, to define an interface to ST-II [9], a new class for the CM data transfer could be derived from the existing class interfacing to UDP. Changes would be minimal.

Communication systems need to keep the number of copy operations low [56]; especially CM streams with their high data rates are critical in this respect. Within MTP, CM data is not copied at the sending side and only once at the receiving side. Only references to CM data are exchanged between modules.

However, we still need three copy operations within CM Client and CM Agent:

1. Compressed data from network to memory
2. Decompressed frames to shared memory
3. Decompressed frames to graphics adapter.

Other copy operations performed by the operating system or by the X server decrease the overall system performance but are beyond our control. Other research groups are working on minimizing copy operations within the communication system [16], especially for TCP/IP [11], and on improving the operating system [21] to handle CM streams as efficiently as possible.

[2]URL= ftp://pi4.informatik.uni-mannheim.de/pub/XMovie

[3]We wrap a C++ class around the C implementation of the decoders. By that all decoders have now the same interface and can easily integrated in XMovie between MTP and display.

XMovie has been implemented in C++ on four different platforms simultaneously in order to continuously verify portability and extensibility. These platforms are:

- DEC AXP under OSF/1 V1.3
- DECstation under Ultrix V4.3
- Sun SPARCstation under Solaris 2.3
- IBM RS/6000 under AIX V3.2

Only system calls available on all platforms are used. Currently we are extending our system to support a great variety of image and movie compression formats such as Motion JPEG, H.261, and MPEG. Software encoders and decoders are available for all formats (e.g. [45, 57]). Version II of XMovie became operational in September 1994.

5.3 Performance

Our current testbed consists of two workstations, one DECstation 5000/133 and one SPARCstation 10/30, both attached to a local FDDI ring. Both machines are used as they came out of the box, i.e. no tuning of kernel buffers, priorities etc. was performed. During our measurements, besides CM Client or CM server, only normal system activities took place on our machines, and the FDDI ring had to carry only MTP PDUs. In all tests the DECstation is used as the sender and the SPARCstation as the receiver.

The initial version of XMovie runs at 25 frames/s over FDDI, with an image size of 160×128 pixels, and with dynamic adaptation of the color lookup table. The color quality is very high.

The current (re-implemented) prototype of XMovie with integrated software decoders was tested with a set of different movies. Our test movies were generated out of the same sequence of 400 true color frames (24-bit per pixel), consisting of four different sequences of 100 frames. It has therefore three scene changes. For each of the five formats MPEG, H.261, Motion JPEG, XCCC, and 8-bit uncompressed (DeltaCLUT), we generated a color movie in the image size QCIF (176×144). This size was chosen because the H.261 decoder can only process either QCIF or CIF sized images. Values for file size and compression ratio are listed in Table. 6.

Table 6: QCIF Movies

encoding	size [bytes]	compression ratio
24-bit	30412800	1.00
MPEG	406265	0.01
H.261	770144	0.03
XCCC	2654638	0.09
JPEG	9962555	0.33
8-bit	11371312	0.37

The frame rate was measured performing all steps from reading and parsing the movies to decoding and display (Figure 8). In the reverse configuration, i.e. the slower DECstation as receiver, reading and parsing is faster, and decoding is significantly slower, thus limiting system performance.

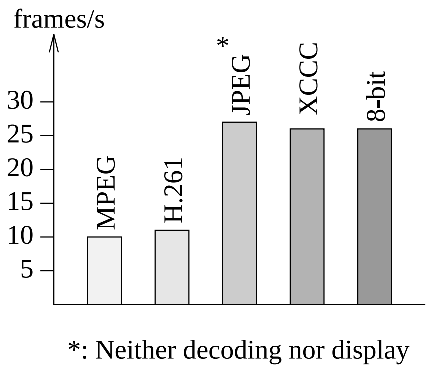

Figure 8: Overall systen performance with QCIF movies

Pure intraframe coded movies have significantly larger file sizes than do interframe coded movies (see Table 6). The decoding of interframe formats is more complex than the decoding of the XCCC coded images. A software decoder for JPEG frames is currently not integrated into our system. Previous tests have shown that with JPEG decoding and dithering in software a frame rate of less than 10 for QCIF can be expected.

The overall system performance of pure intraframe coded movies depends on the amount of data and the encoding used. Reading is faster at the sending side for XCCC coded movies, but the frame rate is reduced at the receiving side by additional computation for decoding and copy operations. JPEG decoding was not measured as mentioned above. The performance difference between H.261 and MPEG coded movies can be explained by the higher complexity of the MPEG algorithm.

Additional processing performed at the receiving side and system limitations reduce the frame rate further.[4] Above all copy operations and operating system kernel interactions are critical in this respect.

6 Conclusion and Outlook

The XMovie system is a test bed for digital image transmission and presentation in a network of UNIX workstations. It is entirely based on standard hardware and uses standard network technology and standard graphics adapters with color lookup tables.

We have presented several new concepts in the context of XMovie. The first is Delta-CLUT, an algorithm for the continuous adaptation of a color lookup table for digital movies. It allows the presentation of films with excellent color quality using standard color lookup table graphics adapters. Run time performance is very good, but the compression ratio is low.

For the presentation of movies on the workstation screen, a new extension to the X window system was presented. It provides native movie windows, with the X server un-

[4]Also take into account the overhead caused by the instrumentation of the code.

derstanding what a movie is. It was shown that this extension can be implemented and used without major problems. An arbitrary number of movie windows can be handled. Experience with XMovie has shown that the presentation of digital movies in a network of standard UNIX workstations without video hardware is feasible.

Experience has shown that standard Ethernet technology and TCP/IP protocols are not suitable for movie transmission. In particular the Sliding Window flow control and the Go-Back-n error recovery scheme in TCP are inappropriate. The transmission of digital movies will require new protocols which are at the same time isochronous, i. e. have a very low delay jitter, and provide high throughput. To avoid retransmission we have developed AdFEC, an adaptable forward error correction method integrated into the application layer of XMovie.

Finally we have presented performance measurement results for DeltaCLUT, AdFEC and the XMovie prototype with different compression formats.

Currently we are extending XMovie to support pure audio streams as well as combined audio/video streams, e.g. MPEG system streams. As soon as we have direct access to an ATM network we will port XMovie to run over ATM. Further, we will add a multicast transport interface to MTP.

References

[1] Anderson, D. P., Delgrossi, L., and Herrtwich, R. G. Process Structure and Scheduling in Real-Time Protocol Implementations. In *Kommunikation in Verteilten Systemen,* GI/ITG-Fachtagung, Mannheim (1991), W. Effelsberg, H. Meuer, and G. Müller, Eds., Informatik-Fachberichte 267, Springer-Verlag, Berlin Heidelberg, pp. 83–95.

[2] Apple Computer, Inc. *Inside Macintosh: QuickTime.* Addison-Wesley Publishing Company, Reading, MA, Mar. 1993.

[3] Berners-Lee, T., Cailliau, R., Luotonen, A., Frystyk Nielsen, H., and Secret, A. The World-Wide Web. *Communications of the ACM 37*, 8 (Aug. 1994), 76–82.

[4] Biersack, E. W. Performance Evaluation of Forward Error Correction in ATM Networks. *Computer Communication Review 22*, 4 (Oct. 1992), 248–257.

[5] Burger, J. *The Desktop Multimedia Bible.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1993.

[6] Campbell, A., Coulson, G., Garcia, F., and Hutchinson, D. A Continuous Media Transport and Orchestration Service. *Computer Communication Review 22*, 4 (Oct. 1992), 99–110.

[7] Campbell, G., DeFanti, T. A., Frederiksen, J., Joyce, S. A., Leske, L.-r. A., Lindberg, J., and Sandin, D. J. Two Bit/Pixel Full Color Encoding. *Computer Graphics* (1986).

[8] Casner, S. Are You on the MBone? *IEEE MultiMedia 1*, 2 (Summer 1994), 76–79.

[9] Casner, S., Lynn, C., Park, P., Schroeder, K., and Topolocic, C. Experimental Internet Stream Protocol, Version 2 (ST-II). Network Working Group, Request for Comments 1190, University of Southern California, CA, 1990.

[10] Video Codec for Audiovisual Services at p$\times$64 kbit/s. Recommendation CCITT H.261, 1990.

[11] Dalton, C., Watson, G., Banks, D., Calamvokis, C., Edwards, A., and Lumley, J. Afterburner. *IEEE Network 7*, 4 (July 1993), 36–43.

[12] de Prycker, M. *Asynchronous transfer mode: Solution for Broadband ISDN.* Ellis Horwood Limited, 1993. 2nd ed.

[13] Deering, S. Simple Internet Protocol. *IEEE Network 7*, 3 (May 1993).

[14] Delgrossi, L., Halstrick, C., Herrtwich, R. G., and Stüttgen, H. HeiTP – A Transport Protocol for ST-II. In *Proceedings of Globecom 92* (Orlando, Florida, 1992).

[15] Delp, E. J., and Mitchell, O. R. Image Compression using Block Truncation Coding. *IEEE Transactions on Communications* (1979).

[16] Druschel, P., Abbott, M. B., Pagels, M. A., and Peterson, L. L. Network Subsystem Design. *IEEE Network 7*, 4 (July 1993), 8–17.

[17] Federighi, C., and Rowe, L. A. A Distributed Hierarchical Storage Manager for a Video-on-Demand System. In *Proceedings of Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Sci. & Tech.* (San Jose, CA, Feb. 1994). URL= ftp: //tr-ftp.CS.Berkeley.EDU /pub/multimedia/papers/VodsArch-SPIE94.ps.Z.

[18] Ferrari, D., Banerjea, A., and Zhang, H. Network Support for Multimedia. Tech. Rep. TR-92-072, International Computer Science Institute, Berkeley, CA, Nov. 1992.

[19] Green, J. L. The Evolution of DVI System Software. *Communications of the ACM 35*, 1 (Jan. 1992), 53–67.

[20] Hartridge, H. The Visual Perception of fine Details. *Philosophical Transactions of the Royal Society 232* (1947), 519–671.

[21] Herrtwich, R. G. The HeiProjects: Support for Distributed Multimedia Applications. Tech. Rep. 43.9206, IBM European Networking Center, Heidelberg, 1992.

[22] Hild, H., and Pins, M. Variations on a Dither Algorithm. In *EURO GRAPHICS'89* (1989), North-Holland, Amsterdam, pp. 381–392.

[23] Hopper, A. Pandora – an experimental system for multimedia applications. *ACM Operating Systems Review 24*, 2 (Apr. 1990).

[24] Huffman, D. A. A method for the construction of minimum reduncancy codes. *Proceedings IRE 40* (1962), 1098–1101.

[25] Information technology – Digital Compression and Coding of Continuous-Tone Still Images (JPEG). International Standard ISO/IEC IS 10918-1, 1992.

[26] Information technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 MBit/s (MPEG). International Standard ISO/IEC IS 11172, 1993.

[27] Keller, R., and Effelsberg, W. MCAM: An Application Layer Protocol for Movie Control, Access, and Management. In *Proceedings of First ACM International Conference on Multimedia* (Anaheim, CA, Aug. 1993), pp. 21–29.

[28] Keller, R., Effelsberg, W., and Lamparter, B. Performance Bottlenecks in Digital Movie Systems. In *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video* (Lancaster, U.K., Nov. 1993), pp. 163–174.

[29] Keller, R., Fischer, S., and Effelsberg, W. Implementing Movie Control, Access and Management – from a Formal Description to a Working Multimedia System. In *Proceedings of the 14th International Conference on Distributed Computing Systems* (Poznan, Poland, June 1994), IEEE Computer Society, pp. 276–283.

[30] Koegel Buford, J. F., Ed. *Multimedia Systems.* Addison-Wesley, Reading, MA, 1994.

[31] Lamparter, B., Böhrer, O., Effelsberg, W., and Turau, V. Adaptable Forward Error Correction for Multimedia Data Streams. Tech. Rep. TR-93-009, Praktische Informatik IV, Universität Mannheim, 1993. URL= ftp: //pi4.informatik.uni-mannheim.de /pub/techreports/tr-93-009.ps.gz.

[32] Lamparter, B., and Effelsberg, W. X-MOVIE: Transmission and Presentation of Digital Movies under X. In *Network and Operating System Support for Digital Audio and Video,* 2nd International Workshop, Heidelberg, Germany, November 1991 (1992), Lecture Notes in Computer Science 614, Springer-Verlag, Heidelberg, pp. 328–339.

[33] Lamparter, B., Effelsberg, W., and Michl, N. MTP: A Movie Transmission Protocol for Multimedia Applications. In *Multimedia92, 4th IEEE ComSoc International Workshop on Multimedia Communications* (Monterey, CA, Apr. 1992), pp. 260–270.

[34] Lamparter, B., Effelsberg, W., and Michl, N. MTP: A Movie Transmission Protocol for Multimedia Applications. *Computer Communication Review 22*, 3 (July 1992), 71–72.

[35] Leister, W., Müller, H., and Stösser, A. *Fotorealistische Computeranimation.* Springer-Verlag, Berlin, 1991.

[36] Lin, S., and Costello, J. *Error Control Coding.* Prentice Hall, 1983.

[37] Ludwig, L. F., and Dunn, D. F. Laboratory for Emulation and Study of Intergrated and Coordinated Media Communication. *Proceedings of ACM SIGCOMM'87* (Aug. 1987), 283–291.

[38] Luther, A. *Digital Video in the PC Environment.* McGraw Hill Book Company, New York, 1989.

[39] Mackay, W. E., and Davenport, G. Virtual Video Editing in Interactive Multimedia Applications. *Communications of the ACM 32*, 7 (July 1989), 802–810.

[40] McAuley, A. J. Reliable Broadband Communication using a Burst Erasure Correcting Code. *Computer Communication Review 20*, 4 (Sept. 1990), 297–306.

[41] McAuley, D. R. Operating System Support for the Desk Area Network. In *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video* (Lancaster, U.K., Nov. 1993), pp. 3–10.

[42] Neidecker-Lutz, B. K., and Ulichney, R. Software Motion Pictures. *Digital Technical Journal 5*, 2 (1993). URL= ftp: //gatekeeper.dec.com /pub/Digital/DECinfo/DTJ/mm-04.ps.Z.

[43] Nieh, J., Hanko, J. G., Northcutt, D., and Wall, G. A. SVR4 UNIX Scheduler Unacceptable for Multimedia Applications. In *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video* (Lancaster, U.K., Nov. 1993), pp. 35–47.

[44] Ohta, H., and Kitami, T. A Cell Loss Recovery Method Using FEC in ATM Networks. *IEEE Journal on Selected Areas in Communications 9*, 9 (Dec. 1991), 1471–1483.

[45] Patel, K., Smith, B. C., and Rowe, L. A. Performance of a Software MPEG Video Decoder. In *Proceedings of First ACM International Conference on Multimedia* (Anaheim, CA, Aug. 1993), pp. 75–82.

[46] Pins, M. *Analysis and choice of algorithms for data compression with special remark on images and movies (In German).* PhD thesis, University of Karlsruhe, Germany, 1990.

[47] Postel, J. B. User Datagram Protocol. Network Working Group, Request for Comments 768, USC/Information Sciences Institute, Aug. 1980.

[48] Postel, J. B. Transmission Control Protocol. Network Working Group, Request for Comments 793, USC/Information Sciences Institute, Sept. 1981.

[49] Rangan, P. V. Video conferencing, file storage, and management in multimedia computer systems. *Computer Networks and ISDN Systems 25* (1993), 901–919.

[50] Rowe, L. A., Boreczky, J. S., and Eads, C. A. Indexes for User Access to Large Video Databases. In *Proceedings of Storage and Retrieval for Image and Video Databases II, IS&T/SPIE Symp. on Elec. Imaging Sci. & Tech.* (San Jose, CA, Feb. 1994). URL= ftp: //tr-ftp.CS.Berkeley.EDU /pub/multimedia/papers/VodsDB-SPIE94.ps.Z.

[51] Rowe, L. A., and Smith, B. C. A Continuous Media Player. In *Network and Operating System Support for Digital Audio and Video,* 3rd International Workshop, La Jolla, California, USA, November 1992 (1993), Lecture Notes in Computer Science 712, Springer-Verlag, Berlin Heidelberg, pp. 376–386.

[52] Roy, J. U., and Nasrabadi, N. M. Hierarchical Block Truncation Coding. *Optical Engineering 30*, 5 (May 1991), 551–556.

[53] Scheifler, R. W., and Gettys, J. *X Window System: The Complete Reference to Xlib, X Protocol, ICCCM, XLFD.* Digital Press, 1990. 2nd ed.

[54] Schulzrinne, H., and Casner, S. RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force, INTERNET-DRAFT, 1993. URL= ftp: //ftp.internic.net /internet-drafts/draft-ietf-avt-rtp-04.ps.

[55] Shacham, N., and McKenney, P. Packet recovery in High-Speed Networks using Coding. In *Proc. INFOCOM 90,* San Francisco (June 1990).

[56] Svobodova, L. Implementing OSI Systems. *IEEE Journal on Selected Areas in Communications 7*, 7 (Sept. 1989), 1115–1130.

[57] Turletti, T. H.261 software codec for videoconferencing over the Internet. Tech. Rep. RR-1834, Unité de Recherche INRIA-Sophia Antipolis, Jan. 1993. URL= ftp: //ftp.inria.fr /INRIA/publications/RR/RR-1834.ps.Z.

[58] Wayner, P. Digital Video Goes Real–Time. *BYTE* (Jan. 1994), 107–112.

[59] Wolf, L. C., and Herrtwich, R. G. The System Architecture of the Heidelberg Tranport System. *ACM Operating Systems Review 28*, 2 (Apr. 1994), 51–64.

[60] Wolfinger, B., and Moran, M. A Continuous Media Transport Service and Protocol for Real-Time Communication in High Speed Networks. In *Network and Operating System Support for Digital Audio and Video,* 3rd International Workshop, La Jolla, California, USA, November 1992 (1993), Lecture Notes in Computer Science 712, Springer-Verlag, Berlin Heidelberg, pp. 169–182.

[61] Yu, P. S., Chen, M.-S., and Kandlur, D. D. Grouped sweeping scheduling for DASD-basd multimedia storage management. *Multimedia Systems 1*, 3 (1993), 99–109.

[62] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. RSVP: A New Resource ReSerVation Protocol. *IEEE Network 7*, 5 (Sept. 1993).