REIHE INFORMATIK
7/93

**Performance Bottlenecks in Digital Movie Systems**

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

Erschienen in:
Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Seiten 163-174, Lancaster, U.K., November 1993

# Performance Bottlenecks in Digital Movie Systems

Ralf Keller, Wolfgang Effelsberg and Bernd Lamparter

Praktische Informatik IV
University of Mannheim
68131 Mannheim, Germany
{keller, effelsberg, lamparter}@pi4.informatik.uni-mannheim.de

**Abstract.** Digital movie systems offer great perspectives for multimedia applications. But the large amounts of data involved and the demand for isochronous transmission and playback are also great challenges for the designers of a new generation of file systems, database systems, operating systems, window systems, video encoder/decoders and networks. Today's research prototypes of digital movie systems suffer from severe performance bottlenecks, resulting in small movie windows, low frame rates or bad image quality (or all of these). We consider the performance problem to be the most important problem with digital movie systems, preventing their widespread use today. In this paper we address performance issues of digital movie systems from a practical perspective. We report on performance experience gained with the XMovie system, and on new algorithms and protocols to overcome some of these bottlenecks.
**Keywords:** Multimedia systems, digital video, realtime networks, performance measurement

## 1 The XMovie Testbed

The XMovie system is an experimental prototype developed at the University of Mannheim [9]. 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. Therefore our system is highly portable.

The system has been operational for about two years and was ported to DECstation 5000, to Sun SPARCstation 10 and to IBM RS/6000. The movies are currently transmitted over Ethernet or FDDI, using IP and UDP and our own Movie Transmission Protocol [10] on top of these. The movies we use for experimentation are digitized videos and computer–generated films.

Three alternative display architectures for movies have been reported in previous work: The first one uses a blue–box architecture like Pandora's box [6].

The special hardware needed limits flexibility, and makes integration with other windows difficult; it also limits the number of movie windows. The second alternative shows a movie as a fast series of single X Window images, usually implemented using shared memory. In this approach the X client has to send a request for each image to the X server. The MPEG player of UC Berkeley uses this approach [15]. Its main disadvantages are that synchronization is up to the application, that the X client and X server must be co–located to gain highest speed, and that multiple movies create considerable overhead. The third alternative is the XMovie approach, an extension to the window system, with a Movie Transmission Protocol between movie server and movie client.

We have extended the X server to process several movie orders from X clients simultaneously [7]. The X server reads the frames sent by a movie server directly from the net and displays them in a window (see Fig. 1).

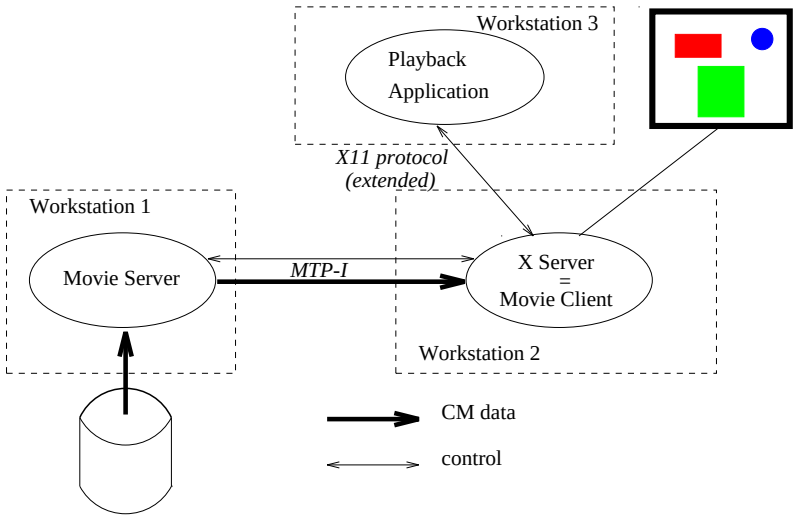


**Fig. 1.** XMovie system architecture

The main advantage of our approach is that the X server can schedule the display of images according to its own requirements, i.e. without deterioration of other X clients, and to those of the movie, i.e. frame rate and isochronous flow. If the X server is heavily loaded it finds a compromise between the two.

The current implementation is not designed to support live video. This will be added in an improved version of the system (see Sec. 4).

Our paper is organized as follows. In Section 2 we present bottlenecks identified in our system. Section 3 describes the optimization of algorithms and protocols. The restructuring of our system for new requirements and higher performance is outlined in Section 4. Section 5 concludes the paper.

## 2 Finding Bottlenecks

In order to find bottlenecks, we start with an analysis of the path the movie takes from the source to the sink: the file system of the movie server, its host bus, its network adapter card, the network itself, the receiver's network adapter card, its host bus, memory, CPU and the graphics adapter card.

The experimental results reported in this section are based on the hardware/software environment shown in Fig. 2. The DECstation 5000/120 has 16 MByte of main memory and Ethernet and CDDI (FDDI over copper wires) network adapter cards. The SPARCstation 10/30 has 32 MByte of main memory and Ethernet and FDDI network adapter cards. The operating systems are ULTRIX and SunOS. The window system is X11 Rel.5. The movies are stored on local hard disks. All clients and servers run on both platforms. Thereby we can measure performance in both directions.

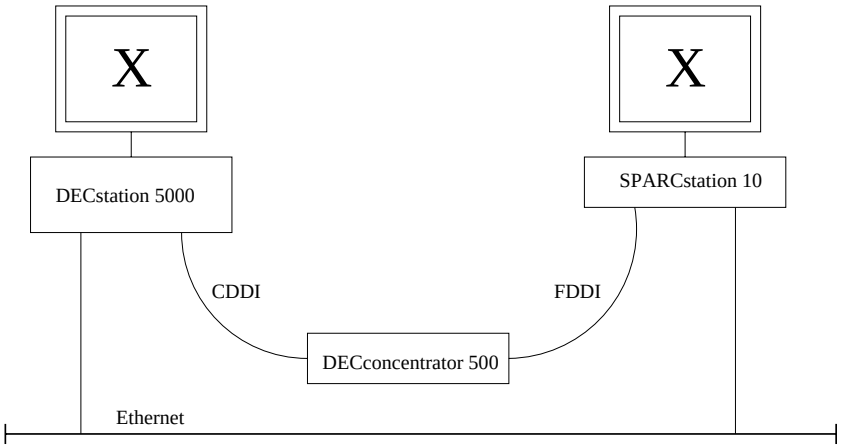


**Fig. 2.** Hardware/software environment

A multimedia file system requires storage and retrieval of continuous media, and efficient manipulation of huge media objects. Our digital mass storages are controlled by the UNIX file system, which fails in both respects. Unix operating systems such as ULTRIX or SunOS do not give real–time guarantees due to their resource reservation and scheduling algorithms (a multimedia enhancement to the UNIX operating systems is described in [5]). A special multimedia file system can also help to solve these problems [16]. If we retrieve more than one movie simultaneously from hard disk, the frame rate of each movie is lower and the jitter between consecutive frames is higher than if we were to retrieve each movie from a separate disk. This is because the seek and latency times between block reads for two different movies are much higher than between reads of consecutive blocks

of a movie. This effect is not only measurable but also audible: the mechanics of our hard disk work very noisily.

In addition, the storage device technology has to be improved. Consider the case where a two–hour movie has been stored on a hard disk in such a manner that it can be retrieved very efficiently in real time, i.e. a constrained block allocation has been performed. What happens if a second, third, etc. request for this movie arrives every 10 minutes (a typical scenario for a video–on–demand service)? One solution to this problem might be multiple disk arms.

Another interesting bottleneck is the network adapter. Our first implementation of the XMovie system was limited by very slow Ethernet adapters with only 2 MBit/s throughput. Today we use FDDI, and our FDDI adapters are so fast that the receiver's CPU (in combination with the operating system) is now the main bottleneck. It cannot process outgoing or incoming network data at the FDDI rate.

The FDDI data transmission rate is 100 MBit/s. The effective sustained data rate at the data link layer can be well over 95 % of this peak rate [17]. We measured a maximum data rate of 20.9 MBit/s at the transport layer with our current equipment (see Sec. 3.2).

Due to packet size limitations of the network (<= 9000 bytes in our environment), a movie frame cannot be transmitted in a single packet. Therefore the X server has to read several packets for each frame from the network. This introduces considerable additional overhead into the X server. The number of packets processed is limited by the speed of the X server dispatch loop;[1] in each cycle only one packet of each movie stream is processed.

Our measurements have shown that the X server is not able to read more packets from the network than are needed to provide a frame rate of 25 frames/s with an image size of $160 \times 128$ pixels (= 4.5 MBit/s), which is relatively small. If we transmit larger images, the frame rate decreases as a linear function of the increase of picture packets.

Now we have identified a great number of *potential* bottlenecks, only one of which, however, can be the limiting bottleneck at a given time. Since we are interested in developing digital movie systems without special hardware and we do not get vendor source code to improve operating or communication systems, we are only able to widen and remove a small number of bottlenecks: transmission of continuous media data (CM data) from the sender through the network to the receiver and to the display; more specifically everything which can be improved on this data path in our own client and server code.

## 3 Optimizing Algorithms and Parameters

### 3.1 Forward Error Correction

If images, fragmented into packets, are transmitted over an unreliable network, some packets can be lost by the network. If essential parts of an image are lost

[1] In this loop all incoming events such as mouse klicks and client requests are processed by the X server.

or incomplete, e.g. the color lookup table or an I–frame of an MPEG stream, visually very disturbing effects appear.

We have implemented our own CM stream protocol with rate–based flow control and forward error correction (FEC) [8]. FEC allows recovery from packet loss, the main error source in modern networks, without retransmission, thus preserving isochronity. Our FEC scheme can introduce an adaptable amount of redundancy and can thus be tailored to the importance of particular pieces of data in the stream.

Let us consider an example where we want to send two packets $P_1$ and $P_2$ and a redundancy of 100% is acceptable (see Fig. 3). Then two additional packets $P_1 \circ P_2$ and $P_1 \bullet P_2$ can be calculated and sent with the original packets. If at least two of the four packets arrive at the receiver side, the original two packets can be reconstructed (in our example $P_1$ and $P_1 \circ P_2$ are lost by the network). It can be shown that our FEC increases the arrival probability significantly. To decrease the loss probability in the event of burst errors, packets can be dispersed across the data stream. More details about our FEC scheme can be found in [8].

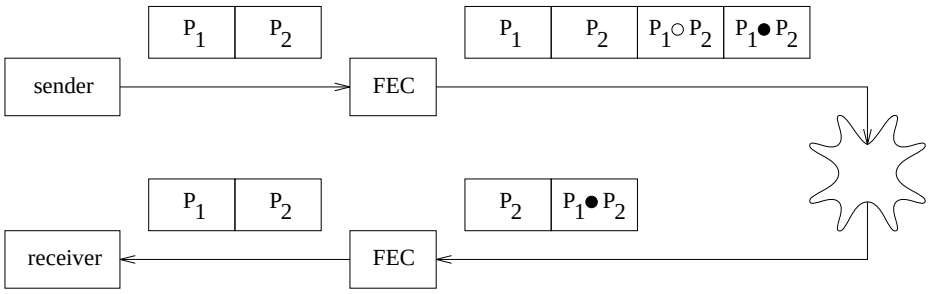


**Fig. 3.** Principle of forward error correction for packet loss

Since additional traffic can congest the network and lead to even higher loss, only important data parts should be protected. In a first implementation, FEC was used to protect color lookup tables (CLUTs) of series of 8–bit color images. These CLUTs are dynamically adapted during movie playback, and CLUT loss would decrease the quality significantly (more than a single frame loss) [10]. One of the main points of interest in our actual work is to investigate how different movie compression formats can be protected. In Motion JPEG (that is, the encoding of video as a sequence of JPEG frames [20]), every $n$th frame (e.g $n = 3$) can be protected to guarantee a certain quality level. MPEG streams [11] can be protected by inserting redundant information for every I–frame. If the I–frame interval is too long, then P–frames can be protected in addition.

Our scheme can be dynamically adapted to network load and error probability. This is much more flexible than the use of channels with different priorities for different frame types as suggested in [3].

We have also investigated the performace implications of FEC in software. All computations were very efficiently implemented with very few machine instructions using table lookup. Protection of only important parts of a CM stream minimizes overhead at the sending side. In addition, packet loss is minimized by rate–based flow control and careful adaptation of system parameters (see next section). Therefore the receving side rarely has to perform FEC.

### 3.2 Packet Size and Throughput

Several experiments have shown that a good algorithm to determine appropriate values for packet size, interpacket gap, and burst size can maximize throughput, and minimize error frequency and delay jitter on our local area networks. On Ethernet we can transmit up to 9 MBit/s and on our FDDI ring we measured a throuphput of 20.9 MBit/s with a UDP packet size of 8000 bytes and an interpacket gap of 1700 $\mu$s (see Fig. 4).

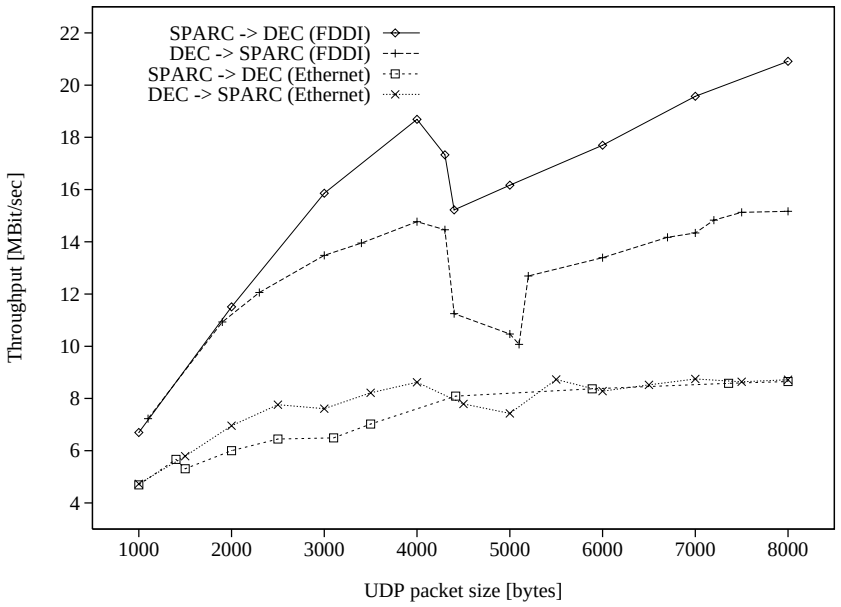


**Fig. 4.** Throughput and packet size

The highest throughput with minimal packet loss freqency ($\leq 0.40\%$) was measured with a SPARCstation 10/30 as sender and a DECstation 5000/120 as receiver using UDP over FDDI. In each test run we had to adjust the interpacket

gap to prevent buffer overflow at the receiving side. The "hole" in the graph at 4324 bytes results from the fact that 4352 bytes is the maximum transfer unit (MTU) of FDDI and therefore a UDP packet of size 4324 bytes fits exactly into one FDDI frame. Above this packet size IP has to fragment user data into several IP frames at the sending side and defragment at the receiving side – additional computation that lowers throughput.

In the reverse configuration (DECstation as sender, SPARCstation as receiver), the maximum throughput was only 15.17 MBit/s with an interpacket gap of 0 $\mu$s (receiving is faster than sending and the DECstation is much slower than the SPARCstation). Sender and receiver performed no work for other processes in these test runs. When both were loaded with additional work, e.g. reading from hard disk or decoding, throughput in all configurations decreased significantly.

To improve the throughput of the communication subsystem, Dalton et al. have shown that with an efficient implemention using a network–independent card to provide the services that are necessary for so called single–copy protocol stack for TCP/IP (i.e. only one copy operation from network interface buffer to application program buffer), a significant speedup can be achieved, especially if the packet size is large enough. They report a TCP/IP throughput of up to 210 MByte/s on HP 9000/730 [2].

In a summary of our tests we conclude that it is better to build large user data packets in the application layer because IP performs the fragmentation and defragmentation anyway and does so more efficiently. Especially in a heterogeneous environment the PDU size and interpacket gap for each pair of senders and receivers has to be determined exactly for optimal throughput and minimal error frequency. Currently we use static parameters which produce suboptimal results but work in each configuration. We are still seeking an efficient method to determine these parameters at runtime.

### 3.3 Flow Control and Jitter Elimination

Another problem was the introduction of rate–based flow control and jitter elimination in the XMovie system. Because every functional unit in the receiving machine (e.g. FEC and decompression) can add jitter to the CM stream, we decided to place these functions as close to the sink as possible (this is also suggested in [14]) and to remove jitter in memory buffers. We separated the receive function and display function into two processes which exchange CM data through shared memory (SHM), one image per SHM segment, with a sophisticated configuration of semaphores to synchronize mutual access. We discovered that the limited number of shared–memory segments available per process and operating system overhead due to the extensive use of semaphores prevented delay jitter control at higher frame rates (see [13] for similar experiences).

We have developed a new queueing scheme implementable with simple operating system primitives. Now it is possible to create longer queues of frames, thus enabling delay jitter control at higher rates. An example is shown in Fig. 5. This architecture also facilitates the integration of software decoders (i.e. for JPEG or MPEG) into the receiving process.

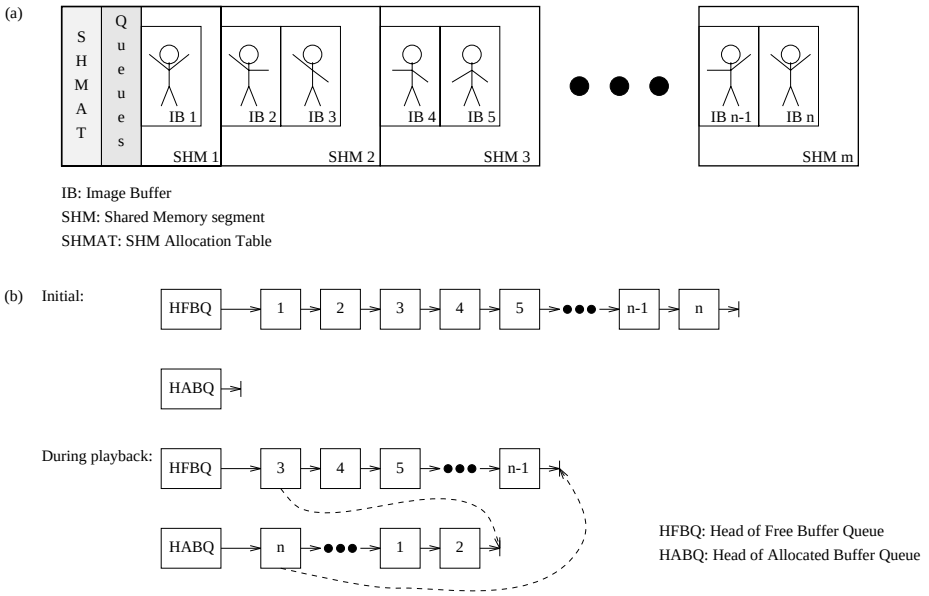


**Fig. 5.** Shared memory queueing scheme

Our goal is to build up queues of images in SHM. We therefore attach several SHM segments and treat them as one continuous buffer with one restriction: allocation of memory across segment boundaries is not allowed, i.e. each image lies entirely within one SHM segment. All administration information is kept in the SHM allocation table (SHMAT, see Fig. 5a). Now we can define two queues of image buffers (IBs): one for the sender (free IBs) and one for the receiver (allocated IBs). The initial state of both queues is shown in Fig. 5b. During playback, IBs from the top of one queue are appended to the other queue according to the following algorithm:

1. If this is the last element in my queue, lock other queue
2. Take IB from head of my queue
3. Read/write contents
4. Attach IB to other queue
5. If this was the last element in my queue, write correct end-of-queue information to head of other queue and unlock other queue

With this buffer management scheme we only have to synchronize if one of the queues reaches its limit. The synchronization was implemented using a simple lock flag.

This queueing scheme can also be used to implement selective frame drop and duplication efficiently. These are required to achieve intermedia synchronization with audio streams as suggested in [12]. Hereby the movie client, which reads

frames from the network and writes them into shared memory, can recover from frames lost in the network by duplicating a successfully received frame. The movie client simply inserts a new IB reference into the allocated buffer queue, referring to an image already stored in shared memory. In case of temporary overload, the movie agent can drop frames simply by skipping step 3 of the above–mentioned algorithm.

## 4 Restructuring our System

To combine the advantages of the old XMovie approach and the jitter elimination with shared memory, we have redesigned our X Windows extension. Our new system architecture is shown in Fig. 6.

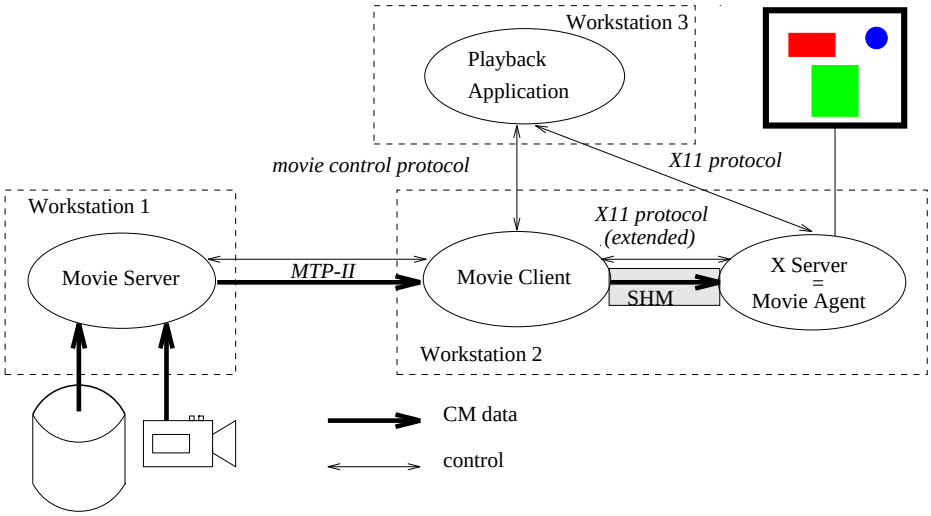


**Fig. 6.** New system architecture

We separated the movie receiving and movie display functions into two processes: movie client and movie agent. The movie client reads the movie stream from the network, performs FEC and optionally decompression in software, and writes single frames into shared memory. The movie agent resides within the X server as an extension to the X code and reads frames at a specified rate out of shared memory. Both communicate through an extended X protocol similar to the old XMovie extension [9] with the following new functions:

**XSOpenStream():** Opens a SHM movie stream and allocates resources (SHM segments and memory)

**XSAllocateImageBuffer():** Allocates an image buffer in the free buffer queue
**XSAppendImageBuffer():** Appends an image buffer to the allocated buffer queue
**XSStartStream():** Starts a movie stream in a window; the movie agent starts displaying
**XSUpdateStream():** Changes the frame rate during playback
**XSStopStream():** Stops a running stream
**XSCloseStream():** Frees all resources allocated for the movie stream

The X server is still able to process several movie orders from movie clients simultaneously. All connections have to share available system resources, in particular SHM segments.[2] But this is not a severe restriction since we do not believe that more than one or two movie playbacks will be active simultaneously, and in live video only a small number of frames can be buffered since live video requires low end-to-end delay. The amount of buffering could be reduced if the underlying communication system supported real–time traffic [1].

The main differences between the architectures described in [18] and ours is that movie client and movie agent are running independently of each other during the playback of a movie, and that we use software decompression. The movie agent receives only a start order and performs the reading and displaying of single frames. The movie client writes frames to shared memory and does not have to inform the X server to display each frame each time. This simplifies the movie client. If both movie client and movie agent are able to fulfill their duty, i.e. to deliver and to display the frames at the requested rate, the system works very smoothly.

We had to take care to keep the number of copy operations low. We still need at least three copy operations within movie client and movie 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 [4], especially for TCP/IP [2], and improving the operating system [5] to handle CM streams as efficiently as possible.

## 5 Summary

We summarize that careful analysis of bottlenecks, provision of new algorithms such as forward error correction, and computer–supported optimization of all

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

system parameters improves the performance of digital movie systems considerably. As an example we have shown the influence of the TPDU size parameters on throughput. We are convinced that digital movie systems can be built without special hardware. XMovie now runs at 25 frames/s over FDDI, with an image size of $160 \times 128$ pixels, and with dynamic adaptation of the color lookup table.

Although specialized hardware, e.g. a hardware decoder for JPEG or MPEG, can help provide better performance in the short term, its usage makes the system inflexible and not portable. Our approach is to use software solutions wherever possible. Our experience in implementing the XMovie system helped us enormously to increase the portability of our system, which runs on DECstations, SPARCstations and IBM RS6000 (and soon on DEC AXP).

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. [15, 19]). We expect that in a generation or two, computer systems will be fast enough for the multimedia challenge. And we are continuing to remove bottlenecks so as to improve our system.

## Acknowledgement

The authors would like to thank Rainer Buchwald, Christoph Wortig, and Oliver Kohnke who did a lot of programming and long test series to prove our assumptions. We also thank our reviewers whose remarks helped us to improve the paper.

## References

1. David D. Clark, Scott Shenker, and Lixia Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. *Computer Communication Review*, 22(4):14–26, October 1992.
2. Chris Dalton, Greg Watson, David Banks, Costas Calamvokis, Aled Edwards, and John Lumley. Afterburner. *IEEE Network*, 7(4):36–43, July 1993.
3. Luca Delgrossi, Christian Halstrick, Dietmar Hehmann, and Ralf Guido Herrtwich. Media Scaling for Audiovisual Communication with the Heidelberg Transport System. In *Proceedings of First ACM International Conference on Multimedia*, pages 99–104, Los Angeles, CA, August 1993.
4. Peter Druschel, Mark B. Abbott, Michael A. Pagels, and Larry L. Peterson. Network Subsystem Design. *IEEE Network*, 7(4):8–17, July 1993.
5. Ralf Guido Herrtwich. The HeiProjects: Support for Distributed Multimedia Applications. Technical Report 43.9206, IBM European Networking Center, Heidelberg, 1992.
6. Andy Hopper. Pandora – an experimental system for multimedia applications. *ACM Operating System Review*, 24(2), April 1990.
7. Ralf Keller, Bernd Lamparter, and Wolfgang Effelsberg. Erweiterung von X für digitale Filme. *PIK – Praxis der Informationsverarbeitung und Kommunikation (in German)*, 15(4):196–204, October 1992.

8. Bernd Lamparter, Otto Böhrer, Wolfgang Effelsberg, and Volker Turau. Adaptable Forward Error Correction for Multimedia Data Streams. Technical Report TR-93-009, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1993. URL= ftp: //pi4.informatik.uni-mannheim.de /pub/techreports/tr-93-009.ps.gz.
9. Bernd Lamparter and Wolfgang Effelsberg. 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, Lecture Notes in Computer Science 614, pages 328–339. Springer-Verlag, Heidelberg, 1992.
10. Bernd Lamparter, Wolfgang Effelsberg, and Norman Michl. MTP: A Movie Transmission Protocol for Multimedia Applications. *Computer Communication Review*, 22(3):71–72, July 1992.
11. Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM*, 34(4):46–58, April 1991.
12. Tom D.C. Little and F. Kao. An Intermedia Skew Control System for Multimedia Data Presentation. In *Network and Operating System Support for Digital Audio and Video,* 3rd International Workshop, La Jolla, California, USA, November 1992, Lecture Notes in Computer Science 712, pages 130–141. Springer-Verlag, Berlin Heidelberg, 1993.
13. Roger Needham and Akira Nakamura. An Approach to Real-time Scheduling but it is Really a Problem for Multimedia? In *Network and Operating System Support for Digital Audio and Video,* 3rd International Workshop, La Jolla, California, USA, November 1992, Lecture Notes in Computer Science 712, pages 32–39. Springer-Verlag, Berlin Heidelberg, 1993.
14. Craig Partridge. Isochronous Applications Do Not Require Jitter-Controlled Networks. Network Working Group, Request for Comments: 1257, 1991.
15. Ketan Patel, Brian C. Smith, and Lawrence A. Rowe. Performance of a Software MPEG Video Decoder. In *Proceedings of First ACM International Conference on Multimedia*, pages 75–82, Los Angeles, CA, August 1993.
16. P. Venkat Rangan and Harrick M. Vin. Designing File Systems for Digital Video and Audio. *ACM Operating System Review*, 25(5):81–94, October 1991.
17. Floyd E. Ross, James R. Hamstra, and Robert L. Fik. FDDI – A LAN among MANs. *Computer Communications Review*, 20(3):16–31, July 1991.
18. Lawrence A. Rowe and Brian C. Smith. A Continuous Media Player. In *Network and Operating System Support for Digital Audio and Video,* 3rd International Workshop, La Jolla, California, USA, November 1992, Lecture Notes in Computer Science 712, pages 376–386. Springer-Verlag, Berlin Heidelberg, 1993.
19. Thierry Turletti. H.261 software codec for videoconferencing over the Internet. Technical Report RR-1834, Unité de Recherche INRIA-Sophia Antipolis, January 1993. URL= ftp: //ftp.inria.fr /INRIA/publications/RR/RR-1834.ps.Z.
20. Gregory K. Wallace. The JPEG Stillpicture Compression Standard. *Communications of the ACM*, 34(4):30–44, April 1991.

This article was processed using the LaTeX macro package with LLNCS style