[CFKS92] *Andreas Cramer, Manny Farber, Brain McKellar Ralf Steinmetz;* **Experiences with the Heidelberg Multimedia Communication System: Multicast, Rate Enforcement and Performance;** IFIP European High Performance Networking Workshop (ehpn'92), Liege, Belgien, Dezember 1992.

Experiences with the Heidelberg Multimedia Communication System

Multicast, Rate Enforcement and Performance

Andreas Cramer, Manny Farber, Brian McKellar, Ralf Steinmetz

IBM European Networking Center
Broadband Multimedia Communications Department
P.O. Box 10 30 68, 6900 Heidelberg, Germany
E-MAIL: {cramer, mckellar, steinmet} @ dhdibm1.bitnet

The Heidelberg Multimedia Communication System, designed at the IBM European Networking Center in Heidelberg, is intended for high-speed data communications and multimedia data exchange. A first version was demonstrated at CeBIT'92. Challenges encountered during the development included multicast on Token-Ring LAN and rate enforcement. Performance of the prototype is also presented in this paper.

Keyword Codes: H.4.3; H.5.1; C.4
Keywords: Communications Applications; Multimedia Information Systems; Performance of Systems

A **networked multimedia system** is characterized by the generation, storage, processing, communication, and presentation of independent continuous and discrete media in an integrated fashion. The primary challenge for multimedia communications is the full integration of continuous media into the digital world.

The *Heidelberg Multimedia Communication System* (HeiMCS) is designed for high-speed data communications and multimedia data exchange. The *Heidelberg Transport System* (HeiTS) is part of HeiMCS and is aimed at a heterogeneous environment comprising several types of computers with different operating systems and a variety of underlying networks. It uses the *Heidelberg Multimedia Operating System Enhancements* (HeiMOS) and provides its services to the *Heidelberg Multimedia Application Toolkit* (HeiMAT), a multimedia abstraction layer.

The prototype is being developed simultaneously for AIX[1] and OS/2. The goals are to demonstrate the feasibility of current network technology for multimedia applications, to explore the limitations of current protocols, to make appropriate changes and/or enhancements, to prove new implementation concepts, and to exploit and integrate upcoming broadband WANs and LANs.

---

[1]AIX, RISC System/6000, Operating System/2, OS/2, Personal System/2, and PS/2 are trademarks of International Business Machines Corporation. Intel, DVI, and ActionMedia are trademarks of Intel Corporation.

A first version was demonstrated at CeBIT'92. The configuration showed an audio-video distribution application with remote camera control, and demonstrated time-constraint communication over a commercially-available LAN. This paper outlines the main challenges encountered during the development of the system: multicast support and rate enforcement (*i.e.*, to control the traffic that "enters" the communication system). Additionally, the performance of the prototype and its limitations are described [Cra92].

Demonstrated Applications

Reflecting the experience of many presentations of our multimedia work, we conclude that "multimedia needs to be demonstrated", *i.e.*, paperwork is nice to have, prototypes are more convincing, but an application is essential. The ENC's core work concentrates on *multimedia communication* issues, but communications is just part of an integrated multimedia system. Two different application scenarios, each having special requirements, were defined. We call these scenarios *employee information* and *production supervision*.

One part of enterprise communication is **employee information**. To be successful, the information presented must be interesting, current, and readily available. Until now, this has meant that such information has been limited to in-house journals, brochures, and electronic mail. Within IBM an attempt has been made to supplement printed information with audio-visual information. IBM Germany quarterly distributes a video tape called "Tele-Report". An obstacle in "publishing" the video magazine is the expense in terms of time and money associated with distributing video tapes. It was clear to the editors that the long-term viability of the magazine depended on the ability to take advantage of existing resources for distribution, in particular the current investment in networked workstations.

From a technical point of view, audio and video information stored on a file server has to be distributed simultaneously to multiple workstations, preferably using existing computers as well as installed digital networks. This *stored video* (audio and video) technology has many other fascinating applications, *e.g.*, computer-based training, promotional videos, entertainment and remote analysis, where visual information is needed.

Developing control consoles for **production supervision** has always been done with the goal of arriving at highly-integrated solutions, *i.e.*, the control personnel should have all relevant information at their disposal in the most compact form. Previously it was not possible to use integrated video to watch over equipment, resulting in separate displays to monitor production processes. There are advantages to retrofitting video monitoring into existing control applications. Video can be shown in one window while other windows display measurements and alarm indicators along with schematics of the production process. This brings us yet another step closer to complete integration of control technology.

Technically seen, a camera installed in the production area has to be controlled from a remote workstation. The control commands for the camera as well as the live audio and video data should be transmitted over the same digital network. Other scenarios based on the same *live*

video (audio and video) technology include the use of cameras in security systems. The remote control of other devices can be done within the same framework.

Environment

The current goal of our work is to integrate the capabilities needed for multimedia into normal workstations. Already existing resources, such as the CPU and Token-Ring LAN, are exploited and supplemented with new resources needed, *e.g.*, ActionMedia adapters. Crucial to our approach is the use of "off-the-shelve" components. Although we have similar goals as other research projects, for instance Pandora at Cambridge [Hop90; Hop91] and work done at Lancaster [Cou91], we strive to integrate multimedia applications into the existing hardware and software platforms expected to be available in most offices. This is to accelerate the acceptance of multimedia applications in the office environment. The strategy of using commercially available products is also followed by a very similar project at the University of North Carolina [Jef91].

Experiences in advanced protocol engineering decree that multiprocessing, *e.g.*, offered by both OS/2 and AIX, should be exploited. Currently, support for compression and presentation of full motion video is only available under OS/2. Therefore, OS/2 version 1.3 was chosen as platform for our initial prototype development, with porting to version 2.0 in progress. The prototype at CeBIT'92 demonstrates a conceivable configuration for both of the two scenarios described above. It consists of multiple client and two server workstations.

One server is configured with vast disk storage and supplies the clients with stored video programs in compressed, digital form for the employee information scenario. The other server is attached to a remote-controlled camera for production supervision and provides the clients with live video, compressed in real time (both Intel 80386/25 CPUs). The camera is mounted on a swivel-base with two stepping motors allowing remote horizontal and vertical positioning. The focus and zoom can also be controlled remotely. The camera server with attached camera is installed close to the observed object. Both servers distribute their respective video information to the clients by means of a multicast feature.

Client workstations are high-end systems (Intel 80486/33 CPUs). In addition to Presentation Manager applications on the desktop, for example a host terminal emulator, there is also an application showing video information in a window of 256 x 240 pixels. The user can alternatively choose to display the *stored* or *live video* that is always running. The remote camera can be controlled from anyone of the clients. This combination of a video application coexisting with normal Presentation Manager applications demonstrates that audiovisual communication can be embedded into the customary office environment.

For full motion audio and video compression and decompression, *Digital Video Interactive* (DVI) technology is used. With DVI, the data rate required for high-quality interleaved audio and video can be reduced to the data rate of a CD player (about 1.2 Mbit/s). DVI supports both Presentation Level Video (PLV) and Real Time Video (RTV). PLV is an asymmetric compression technique requiring compression to be done at Intel, but allowing decompression

to be performed in real-time. RTV delivers lower video quality, but both compression and decompression can be performed in the workstation in real-time [Gre92].

Token-Ring LAN (16 Mbit/s) is used for the distribution of multiple video streams simultaneously with normal network traffic (*e.g.*, mainframe access and file transfers). The architecture is flexible enough to support a bandwidth management system, initial using MAC priorities [Nag92], to enable delivery of continuous video even over a heavily loaded ring.

System Architecture

Although the choice of OS/2 as prototype platform was strongly influenced by the availability of advanced multimedia adapters, there are other good reasons for using this operating system; important for this prototype was multiprocessing and a graphical user interface. Unfortunately, OS/2 does not support a *real-time environment* (RTE), so for a first version, priority classes were used to distinguish between the RTE and the *non real-time environment* (NRTE). A RTE is needed to support the time constraints of video [Her92]. Most of the communication system and multimedia components must be part of the RTE, while the interface to the user is an example of an NRTE component.

A networked multimedia application **has the following main components:** operating system abstractions, multimedia I/O support, communication system, multimedia abstraction layer (or object oriented mapping layer), and actual application that glue objects together and manage the interface to the user.

The **operating system abstractions** are software libraries to create a superior implementation environment and group operating system dependencies. The goal is to provide better portability for the prototype.

The *Operating System Shield* (OSS) is an abstract interface of functions that are expected to be available in a multiprocessing operating system. The intent is to abstract the concrete underlying operating system, allowing the same interface to be made available on different platforms. Abstractions are provided for basic data types, process synchronization, protection for shared global data objects, inter-process communications and timer support.

The *Buffer Management System* (BMS) provides a high-level interface for the management of buffers and ensures that all processes in the system use buffers consistently. It achieves efficiency by reducing data copying and making buffers available to all entities requiring them. A scatter/gather system, similar to that in the χ-kernel, is used [Hut89]. Powerful utility functions are provided to support protocol processing. The identical interface is supported under both OS/2 and AIX, again allowing for easier porting of protocol implementations.

The **multimedia I/O support** is based on the ActionMedia adapters together with the Audio Video Kernel (AVK). The AVK comprises an interface that supports capturing and compression of audio and video in real-time and allows for decompression and displaying of digital data. Typical operations supported on video streams are play, pause, stop and frame advance.

The AVK uses different threads as part of the RTE to meet the time constrains of audio and video.

The **communications system** can be divided into the high-speed access to network adapters and the actual protocol processing. Network access is handled by a specific device driver. The protocol stack is structured according to the OSI reference model providing Data Link, Network and Transport services. On top of this protocol stack, the Continuous Media Communication Service is implemented to managing the interleaving of audio, video and extra parameters (*e.g.*, compression algorithm and title of the video) in a single stream of data packets. Normally the device drivers execute in their own threads as part of the RTE. For asynchronous execution between different components, parts of the protocol stack also use seperate threads.

So as to allow integration into current LAN infrastructure, IEEE 802.5 Token-Ring LAN is used for the *Physical Layer*. The system is also implemented with a proprietary high-speed adapter exceeding a data rate of 100 Mbit/s. The *Data Link Service* provides an abstraction from the different underlying hardware and protocols, allowing us to easily support the different network architectures in an uniform way. LLC protocols were used in this implementation.

For the *Network Service* a modified X.25 Packet Level Protocol is used. The OSI standardized version was stripped of functions provided at higher layers or that were not appropriate for multimedia communications (error recovery), and then enhanced to support multicasted data streams. The *Transport Service* implements the ISO transport protocols IS-8073 and IS-8602. An "association"-oriented protocol was implemented for the multitarget service to support the distribution of audio-visual data.

The multimedia abstraction layer provides an object oriented interface to control the multimedia data transfer. The essential classes are the *sink*, the *source*, and the *stream*. Once opened, sink and source objects are connected by establishing a stream object. A stream is the abstraction performing, *i.e.*, controlling, the data transfer. Reading from the ActionMedia adapter (through the AVK interface) is modelled as a source object; writing as a sink object. Receiving data from a transport connection is modelled as a source object, and the transmission of data as a sink object. A stream object connects source and sink objects. For example, using the ActionMedia adapter as source object and the communication system as sink object, will result in data being read from the ActionMedia adapter and then transmitted. Stream objects thus define the interaction between source and sink objects. Other parts of the system are also modelled as sources and sinks. For instance, an audio-video stored server (AVSS) used to read time-critical data from disk can be represented as a source object at the abstraction layer.

In general, a source object *generates* data, *e.g.*, by capturing it in real time (live video), by reading it from disk, or by receiving it from the communications subsystem. A sink object *consumes* data, *e.g.*, by displaying it on the screen or by transmitting it. This makes it easy to build different types of applications by linking various source and sink objects together using stream objects.

Multimedia servers and clients are configured to provide the correct interaction between various components already described. All the basic components were modelled as source and sink objects. Different configurations could be build by connecting source and sink objects with a stream object.

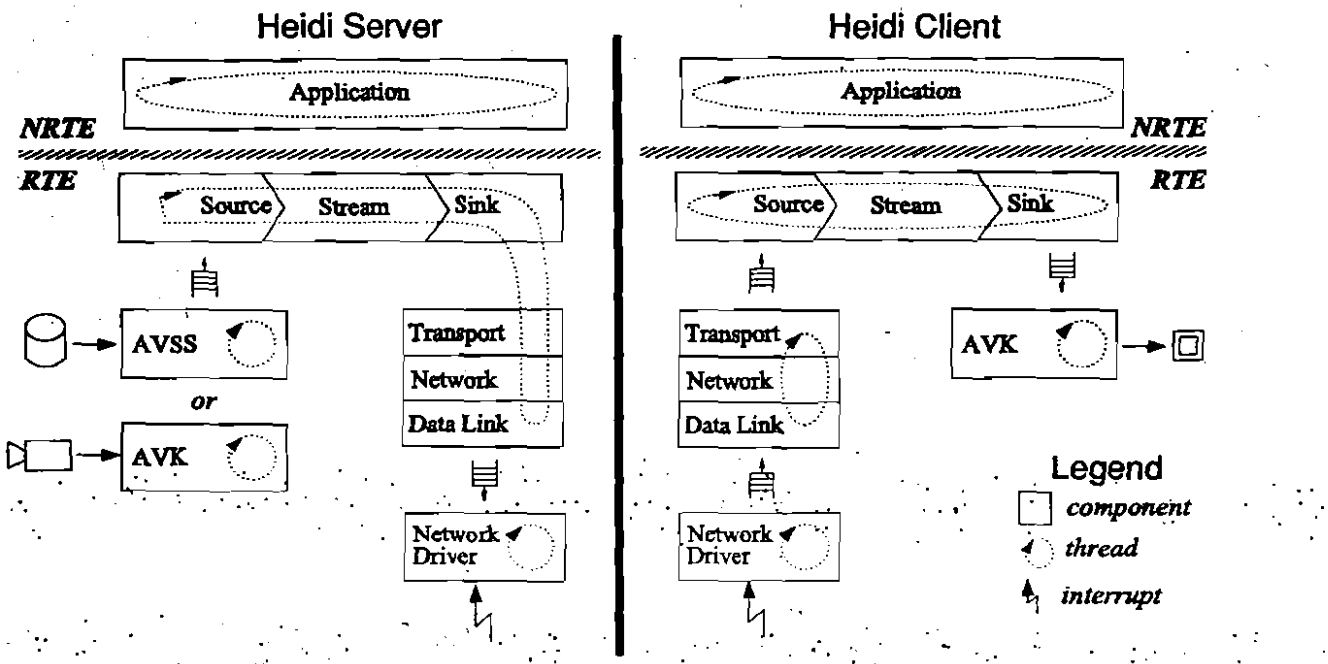

Figure 1 HeiDi, continuous media sender and receiver

The *audio-video file server* uses the AVSS as source to reads video from disk and delivers it to the clients using the multitarget service of the communication system as sink (Figure 1). Data capturing in real-time is performed by the *real-time audio-video capture server*. It has essentially the same components. The only difference is the substitution of the real-time capturing source. The *audio-video presentation client* is also shown in Figure 1. It contains a communication system as source object, connected to an ActionMedia adapter as sink. Note that for local video display, it is also possible to use the prefetcher that reads data from the file system as source directly to the ActionMedia adapter as sink.

The interaction of the overall system configuration is described for a typical configuration comprising a real-time audio-video capture server and a presentation client. At the server, the stream object, executing as a separate thread, retrieves the video data to be transmitted from the source, in this case the AVK doing the real-time capturing. The source object copies the video data into a buffer, as it internally incorporates the AVK product device driver which does not make use of our BMS. If no buffers are available, the stream thread suspends itself for approximately 30 ms (video frames have a *duration* of 33 ms). The same procedure applies to the audio data.

The audio and video packets with an application protocol header are merged into a single *frame*. The frame is subsequently transmitted to the sink, which encapsulates the HeiTS's

transport service interface at the sending station. The transport server segments frames if needed and then, using down-calls, transmits them to the clients. After the protocol processing of all the layers, the resulting PDUs are handed to the network device driver for transmission. It is important to notice that only one thread is used for the multimedia abstraction layer and the synchronous protocol processing for PDUs to be transmitted. The device driver runs in a separate thread, which does the actual interrupt processing and asynchronously transmits PDUs.

At the client, the network device driver executing in its own thread, receives data from the network. The driver enqueues the received PDU to a HeiTS queue determined by the protocol control information. The receive thread passes the PDU through the different protocol layers for processing. A possible reassembly of the PDU is performed. If the current PDU is the last or only PDU of a frame, the total frame will be enqueued to the stream thread. The stream object, executing as a separate thread, dequeues frames and gives them to the sink object, which in this case is the AVK to display the video. If the AVK can not immediately accept a new frame, the stream thread will suspend itself for about 32 ms.

Multicast

The above mentioned applications require many workstations to concurrently receive the same data. We distinguish between two methods of providing such a **multitarget** or **group service:**

- With **multiconnection**, separate connections are established from the sender to each recipient, resulting in the identical data to be sent many times, increasing the network load proportionally to the number of recipients.
- A **multicast** facility uses group addressing to allow data transmitted once over a network to be shared, *i.e.*, received by a group of recipients, implying that no replicated data exists in the network.

The following are considered as **requirements** for a multitarget service:

- For continuous media multitarget associations, zero-reliability is sufficient, because a retransmission is not useful for real-time data. In most cases, retransmission of a lost packet cannot be performed before the deadline of the packet expires [Hug89a].
- Packets sent over a subnetwork should not have any impact on workstations not participating in a multicast association.
- The system must allow different streams to be transmitted simultaneously on the network.
- It should also be possible for a workstation to receive more than one stream concurrently. This is necessary if multiple videos from different conference participants are to be displayed on the same monitor, for cases where one video stream with different audio streams (multilingual soundtrack) is transmitted and gateways must be able to concurrently route multiple streams.

The solution presented in this paper meets all of these requirements, although restricted to Token-Ring LANs.

The Token-Ring adapter interface provides four different addresses for addressing by other stations: an universally administered address (unique per adapter), a locally administered address (user selectable), a group address and a functional address. The adapter can have one address of each type to which it will react. The adapter can also be addressed using a broadcast address.

The *locally administered address* must be unique across the network, and is basically the same as the *universal administered physical address*. These addresses can not be used to support multicasting. The *broadcast address* can be used to send packets to all stations on the network. This allows for a rudimentary implementation of multicasting. However, as all stations on the network are affected by broadcasts, this address is not suitable for a proper multicast facility.

A *group address* is associated with a set of logically related stations. When a packet is sent to a group address, all stations that have this group address set will receive the packet. Group addresses enable the software to easily send a data packet to a selected subset of stations (that have the specific group address set) on the network, without influencing the other stations on the network. It is clear that group addresses are really what is needed to support multicasting. Unfortunately, currently available Token-Ring adapters only support one active group address at anytime. A group address used for multicasting would then only allow for one active stream.

Each adapter can also have one *functional address*. This address is basically a mask that is ANDed against the address in an arriving packet. If not zero, the packet is received. Of the 48 bits in a MAC address, the first two bytes and one bit in the next byte are fixed, leaving only 31 bits for functional addresses. These types of addresses were intended for scenarios where a station can transmit a packet to a specific function, and this function can then be available on any computer. The workstation that offers the function, will set the specific bit in its functional address mask to indicate that the function is available, for example an error report logger. Of the total of 31 bits that can be assigned to functions, 7 bits have already been preassigned to specific functions, 12 bits are reserved and 12 bits are available to the user. We suggest to use the reserved and user defined bits for continuous media data transfer, giving a maximum of 24 unique concurrent streams on the network.

When functional addresses are used for multicasting it gives us exactly what is needed. It does not interrupt none-participating stations, and allows multiple streams at the same time. Unfortunately it limits the number of concurrent streams to a maximum of 24. It can be observed that 12 video streams at 1.2 Mbit/s per stream will saturate a 16 Mbit/s Token-Ring. If the stream-bandwidths are reduced (*e.g.*, for voice only), more streams can be put through the network at any time, but then, we are only trying to develop a realistic test environment. As functional addresses are in fact (special) group addresses, and behave semantically the same, we can define the Token-Ring interface as if we are using group addresses and for the moment implement this with functional addresses.

A known difficulty for a given MAC group is that the sender can not belong to the same group as the targets to ensure that it does not immediately receive packets which it has

transmitted. For a conference system this implies that each sender must use a different MAC group address to transmit frames to the same logical group.

Support for **multicasting in the Data Link Service** is built on the Token-Ring functional addresses as described. The interface provides functions to join and leave a group, and to send and receive packets in a multicast communication mode. The transmission routines are basically the same as the single-target connectionless datagram routines. Both use connectionless data transmission, but with different target addresses.

Multicasting for the Network Service uses a modified X.25 Packet Level Protocol. From the basic X.25 protocol, the connection establishment and data transfer phase are retained with support for multiplexing. Additional packets structures are defined to support (connectionless) multicasting. The network layer also ensures that out-of-sequence packets are discarded, due to the real-time constraints of video data. The ISO Network Service addressing was extended to support group addresses [Jak91]. For efficient communication over interconnected networks, multicasting is used within LANs and multiconnections are used to cross LAN boundaries. For the initial version a static routing scheme was selected that flooded the data stream to all possible nodes [Dee90].

The **Multitarget Transport Service** relies on the multicast support provided by the Network Service and just adds segmenting and rate enforcement. Interestingly, are the modifications made to the service interface to support a multicast service.

The multicast service is based on the idea of a sender *offering* information to a defined group of recipient(s). Any individual recipient may tune in and out as he pleases. At any moment, there may or may not be any active recipients. This situation is analogous to cable-TV programs or a speech where the listeners are free to come and go. The sender will not be notified if a new client is joining or leaving the group (see also [Rav92]). Such an *association* is the combination of the sender, the group of recipients, and a "program" being transferred from the specific sender to the group of recipients.

The service interface allows the user to request an association (sender) or to join an association (receiver). This allows the transport service provider to reject requests if resource reservation fails. Note that at the level of the transport service interface, the only end-to-end interaction are the primitives for data transfer. The service is unreliable in the sense that data is not guaranteed to be delivered to any recipients. It is, however, guaranteed that only complete, error-free and in-sequence PDUs will be delivered, although possibly with missing PDUs in between.

Rate Enforcement

Rate enforcement is part of a flow control technique, *i.e.*, to prevent a fast sender from overrunning a slow receiver. By itself, it guarantees that a sender will not exceed certain rates. This alone is not sufficient — it is also necessary to guarantee that the processing at the receiver(s) will meet (or exceed) the same rates. Resource reservation and real-time scheduling

can ensure this by guaranteeing that the necessary resources (*e.g.*, CPU for protocol processing) will be available in sufficient quantity, when needed. (By contrast, rate control is stronger, ensuring that data will be sent or received at specific times [Kal90; Sid89].)

Rate enforcement is incorporated in the transport service at the sender side, and resource reservation is assumed to be done at the establishment of an **association**, be it multicast or point-to-point [Sch92]. The goal of rate enforcement is to prevent an association from using more than its reserved resources (permitted short-term bursts are taken into account). Using more then the reserved resources could cause degradation of the quality of service on ***guaranteed*** associations in general, not just the one that is misbehaving. Note that an overly constrictive rate enforcement, *i.e.*, one that does not allow the sender to send at its maximum allowable rate(s), is just as bad, since this is just another way of violating service guarantees.

Resource reservation and rate enforcement are unified by the **Quality of Service** (QoS) specification, which must be suitable for both purposes. An approximate upper limit on the usage of various resources (CPU, network bandwidth, memory, etc.) must be derivable from the QoS negotiated at connection establishment time. Vagueness in specification can lead to over-reservation. For instance, a throughput of 1000 bytes per second can be achieved by sending 1000 one-byte packets in a second, imposing a relatively heavy load on the system; or by sending a single 1000-byte packet every second. In the absence of any other QoS specifications, the former must be assumed if a solid guarantee is to be provided, leading to potential resource squandering. The QoS must also be easily enforceable, which calls for a relatively simple specification.

Consider a *QoS specification* having the parameters: data rate (bytes/second), SDU rate (SDUs/second), maximum SDU size (bytes), and maximum workahead (ms). If not all SDUs to be sent are of the maximum SDU size, then a data rate lower than the product of the maximum SDU size and the SDU rate can be specified, reducing the amount of bandwidth that must be reserved. In the case of video compression where large SDUs representing self-contained still frames are followed by a number of smaller delta frames, as with DVI technology, the difference is substantial. If an application generates SDUs in bursts (*e.g.*, after a large block of stored video is read from disk) and wants the transport service to accept a burst for processing, it is asking to be allowed to work ahead of schedule for a short period of time. Workahead is thus viewed as an application requirement.

Workahead is the amount of work that can be queued for processing before old work has been processed to completion. Workahead could be specified differently; which way is more natural depends on the source of the burstiness of the user's data generation. For instance, maximum SDU workahead (number of SDUs) and maximum data workahead (in terms of bytes) could be specified independently. If the user's burstiness is based on the size of buffers from which source data are taken, or limited by where received data are stored, then this specification may be better than one in terms of time units. If the user's burstiness is influenced by the time it takes to perform disk read operations or by the resolution of timers and the scheduling granularity (*e.g.*, timeslice duration) of the operating system, then the time specification is better. Looking at it another way, the workahead represents the work in progress, *i.e.*, SDUs enroute between the local and remote transport user. Such SDUs consume

resources along the line, be it buffer space, network bandwidth, or CPU for protocol processing in the end-system or in intermediate nodes.

The workahead data structure is a FIFO queue of records containing information about SDUs accepted for processing. One end of the queue represents the local transport system, and the other end is the remote transport system, at the point where data indications are enqueued to the transport user. In this abstraction, the queue itself includes processing in the local transport entity, the network layer, and in the remote transport entity. SDUs closer to the front of the queue (where they are removed from the queue) may be in a real, physical sense closer to the remote system. Stored is the time at which the SDU was accepted, size of the SDU, and (conceptually) any other information that may be needed to figure out when processing on a SDU can take place or how much time it will take. Work accepted by the rate enforcer to be processed is added at the back of the queue. Removal from the front of the queue represents processing assumed to have been completed. Note that this is done according to the processing model, without any knowledge of the current state of actual queues in the local or remote systems or in the network layer.

The **processing model** determines when SDUs are removed from the queue and thus, indirectly, whether (or when) new SDUs may be added to the queue. For our processing model, we start with the following assumptions:

- SDUs are sent over a dedicated channel with a capacity equal to the transport data rate of the QoS negotiated at connection establishment time (for simplicity, PDU headers are ignored). The transmission time is thus the SDU size divided by this data rate.
- CPU time for protocol processing in the local and remote end-systems may not be available immediately, but this processing will be completed after at most one SDU period (the reciprocal of the SDU rate). In other words, we assume that the CPU is scheduled such that at least one SDU's worth of processing will be made available after at most one SDU period, as opposed to merely that the average will be met over an indefinite *long*-term.
- Idle time cannot be accumulated. If the network or a CPU sits idle for one minute because it was not given any work to do, it cannot then do two minutes worth of work in the next minute.

We assume two-stage processing, where the two stage processors work in parallel (Figure 2). The first stage processes SDUs at the QoS data rate, and the second stage at the QoS SDU rate. Collectively, the stage processors represent all resources reserved by the transport layer at association establishment time. Although any individual SDU will take either the SDU period or longer to process from sender to receiver, this does not imply that the SDU rate cannot be maintained, since processing by the two stages is done in parallel.

Note that such a two-stage model accurately reflects how work is done in the CeBIT application, where video frames are transmitted as SDUs, and displayed in real-time at 30 frames per second. SDUs are handed over to the network layer, which processes them at a rate dependent on the SDU size. Once the remote application receives them, they are handed to a delivery card which spends a constant 1/30 of a second processing each SDU (frame).

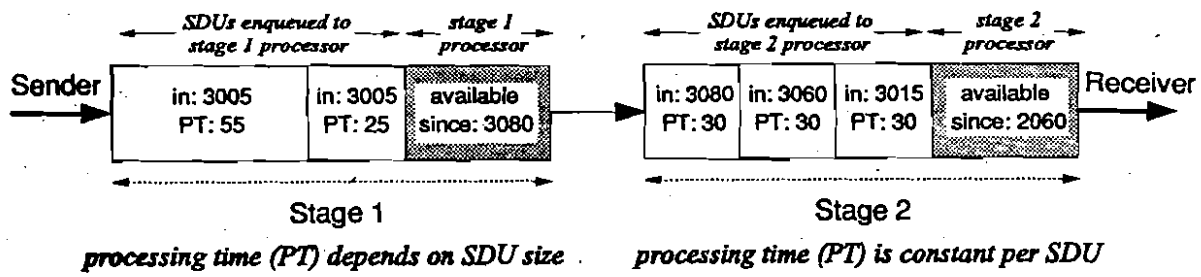

Figure 2 Workahead and processing model

The state of the workahead queue is only important when the question must be answered, of whether the transport service can accept a specific user data request (new work). When a data request is made, the workahead queue is updated according to the processing model based on the current state of the workahead queue, and the current time. The workahead queue lengths are compared with the maximum values to see if the new work can be accepted or not. If so, the new work is added to the back of the workahead queue. If not, then the earliest time at which, if resubmitted, the SDU will be accepted, is calculated and returned to the user to prevent time-consuming polling.

The provision of time services has to be assumed to implement rate enforcement. The basic service needed is that of a **clock**, *i.e.*, a function that returns the current real time. The rate enforcement algorithm must also know its resolution. It is assumed that *sufficiently accurate* real times are available, the time resolution is known, and an upper bound on the lag between the real-world time and the system time is known, which is important in cases where the system time is updated by a software interrupt.

The motivation for rate enforcement is to avoid overuse of resources. Any resources used by the rate enforcement scheme itself, directly or indirectly, should therefore be reserved. This includes, in theory, CPU usage by the rate-enforcement algorithm and resources for processing any SDUs in excess of the *requested workahead*, specified by the user in the QoS, that may be admitted by the rate enforcement. This workahead is called the *enforced workahead*.

Reasons why the enforced workahead may be greater than the requested workahead:

- The workahead must be at least one SDU, otherwise no work will ever be accepted. In particular, in the absence of any other workahead, a maximum-sized SDU (*i.e.*, the maximum specified by the user in the QoS) should be accepted for processing.
- Due to the granularity of the clock, it is possible that work has been done, but is not counted as having been done. Work will only be considered as have been processed when the system time changes. There are two ways to deal with this. One can either accept that the user may get a little less than the workahead he thinks he was guaranteed, or one can allow him perhaps a little more, but reserve the resources for the maximum that will be allowed. The latter strategy was followed. In any case, the workahead must at least be as great as the clock resolution plus the time lag, since SDUs can only be removed from the workahead when the rate enforcer sees the time change, *i.e.*, after every timer tick.

The *enforced* byte and SDU workaheads are the values that should be used in determining resource requirements.

Performance measurements on the prototype

Before starting with the analysis of performance measurements, it is useful to examine in detail the **load** the system is expected to process. In this case, this means characterizing the digitized DVI data-streams which are sent over the transport system as SDUs. A DVI data-stream includes audio as well as video information — including still frames (self-contained images) and delta frames (differential information). Only the video information is considered, although audio information is in the same SDU, since it represents the vast majority of the data.

For the **DVI load** it is useful to examine the differences, with respect to load, between PLV used for stored video and RTV for live video. A sample video clip of an TV interview, 4 minutes 46 seconds long, was recorded using the two different compression methods. The statistics that characterize the digitized frames are listed in Table I. The minimum frame sizes are remarkably similar. However, the maximum frame for the PLV version is more than three times as high as those for RTV. In other video clips, maximum frame sizes of 31020 bytes for PLV and of more than 10000 bytes for RTV have been observed.

Examining the sizes of the PLV frames in chronological order reveals a very regular, cyclic pattern. Usually two small frames, less than 5000 bytes in size, are followed by one large frame having a size between 8000 and 14000 bytes. These are all delta frames, with even bigger still frames interspersed at regular intervals. For (high-quality) RTV frames it is remarkable that there are normally five still frames in the range of 5000 to 8400 bytes followed by one delta with a frame size of 622 to 654 bytes.

Table I Characterization of the DVI load

	PLV	RTV
Number of frames	8600	8600
Minimum [bytes]	738	622
Maximum [bytes]	26384	8334
Mean [bytes]	4968	5366
Median [bytes]	2230	6210
Standard deviation [bytes]	4811	2235
Required bandwidth [Mbit/s]	1.1924	1.2877
Number of deltas	8433	1426
Number of stills	167	7174
Ratio deltas/still	50.50	0.20

From Table I, an average throughput is calculated based on the mean frame size. The throughput for both are about 1.2 Mbit/s. Although the difference is less than 0.1 Mbit/s, an important difference lies between them, namely that the RTV exceeds the 1.2 Mbit/s data rate of a CD-ROM player. It is possible to compress video in real-time that will not exceed 1.2 Mbit/s, if a lower quality video can be tolerated.

The PLV median frame size is less than half its mean frame size, indicating that the differences of the individual frame sizes from the mean value are very high. This can also be recognized by its relatively high standard deviation in comparison to RTV. PLV frame sizes

are very irregular. This irregularity is not desired, as it introduces significant delay jitter, caused by the five frame copy operations necessary for transmission.

On average, for PLV, 50 delta frames are followed by one still frame. For high-quality RTV compression mode, with more still than delta frames, the ratio is about five stills per delta. These figures are important for error handling. If a frame is lost, recovery can only be done at the next still frame. Using PLV compression, a lost frame is followed by an average of 25 delta frames. Thus, on average, an error will be perceptible for a mean time of nearly 1 second. This relation is significantly better with RTV compression, where the average error display time is only a third of a second, which is bearly perceptible.

For **measuring delays within the system**, the *ZM4 monitoring hardware* was used [Hof92]. It is a distributed monitor that allows a large number of computers to be monitored at the same time with one global time of 200ns resolution. The monitor is independent of the observed objects with the only limitation that the observed objects have to be within 1000m of one another.

The focus is on the live video server with one client (using RTV), since delay is only critical in the live video case. The aim is to analyze the delay in the total system from capture server to display client. Additional clients would only have enlarged the data analysis without contributing to the knowledge about the system behaviour. With the hybrid monitoring approach used, it is easy to analyze the internal behaviour of the server and the client, but we also want to measure the frame delay from the camera to the capture process. This cannot be measured directly, because hardware devices, like the DVI adapter, are involved. Therefore we need to measure the delay indirectly.

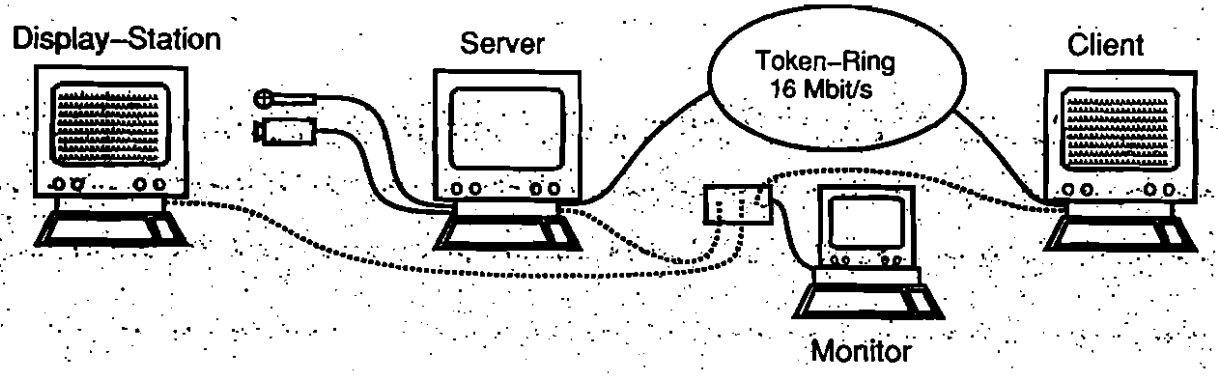

Figure 3 The configuration used for measurement

We assume that a transition of one constant screen pattern recorded by the camera to another constant screen pattern will result in a change of the captured frame sizes. We thus filmed the screen of a third computer also connected to the monitor. When the screen transition occurred, *i.e.*, when the third computer changed the contents of its display, a monitoring event was generated. The first packet from the AVK with a significant difference in frame size indicates the new screen content, allowing us to measure the delay for capturing video. Unfortunately,

no simple way of measuring the delay of the frame on the delivery card until it is displayed on the screen could be found.

In Figure 3 the three observed objects are depicted. On the left is the display station that generates the constant screens for capturing. In the middle is the capture server sending to the client. For measuring, a sequence of screen transitions where generated. It started with an empty screen followed by a screen totally filled with dots (.), then the screen was cleared again, filled with note signs (♪), empty, filled with 'A', and again empty. The capture card was configured such that only still frames should be recorded. This quality is at least as good as that for (high-quality) RTV.

The load is presented in Figure 4. The frame sizes have three significant changes, corresponding to changes of the display's contents. The base line (about 5 Kbytes) is associated with the empty screen. The first change, to 7500 bytes, is correlated with the dots, the second with the note sign and the third with the "A". Because there were only six changes on the screen, only six values for the delay from the camera to the application process could be measured. The values are in the range from 103.6 ms to 135.7 ms. The real values have to be lower than these, because the time between the contents changed on the screen and the first packet with the new contents, is included in the measurement. Interestingly is the (14) peaks down to 622 or 654 bytes caused by delta frames, although only still frames were requested.

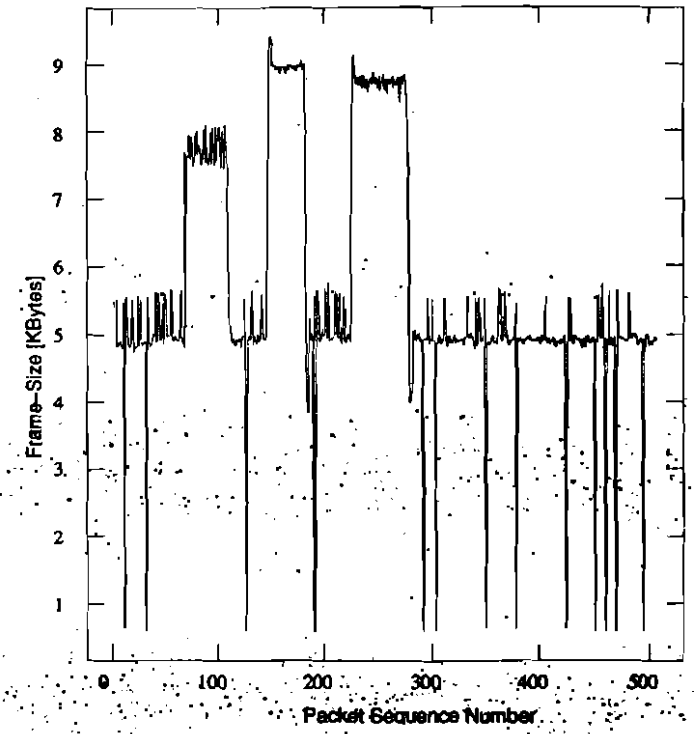

Figure 4 Size of transmitted frames

Figure 5 depicts **the total delay of frame transmission.** This includes the call to the AVK on the server for a new video packet, the assembly to a new frame, transmission over the transport system, and any delay until the stream thread on the client side dequeues the frame from the queue. The graph begins with a peak above the 550 ms level. This is due to the first still frame transmitted being used to initialize the delivery card. This initialization lasts about 500 ms. During this time the client stream thread is unable to process the next frame. The other three delay peaks are correlated with the higher load.

Table II contains significant values of the runtime distributions of the individual parts of the total delay. The first column contains the distribution of the total delay that is shown in Figure 5. The total delay can be divided into three disjoint components, the delay of the

stream thread at the server side, the end-to-end delay of the transport service and the waiting time in the queue to the stream thread on the client side before delivery to the AVK for displaying.

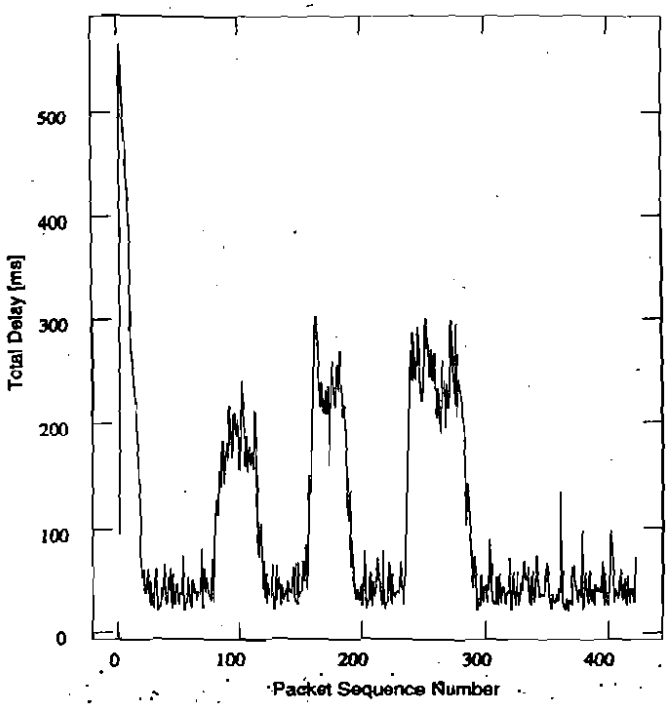

Figure 5 The total delay from application to application

Let's **first consider the delay within the stream thread at the server side.** During this time, the stream thread calls the AVK process for a video packet. If none is available, it sleeps for about 30 ms before retrying. A frame is built and then passed to the transport service. The specific delay distribution parameters are listed in the second column of Table II. The stream thread delay has a large maximum value of about 111 ms. The mean value of 18 ms and the relatively low standard deviation indicate that this maximum value is not representative of the whole distribution. The standard deviation of 14.8 ms depends on the waiting that is done at times.

The **end-to-end transport service delay** is measured as the time from the transport service call at the server side until the frame is enqueued to the stream object, by the transport service on the client side. The measurements were made on an empty Token-Ring and have an unexpected low mean value and median. It indicates that the transmission of the frames do not significantly increase the total delay. Although the delay values for the transport service is low, it is interesting to have a closer look at the delay which is associated with the Token-Ring adapters (last two columns of Table II). Note that segmenting causes the number of Token-Ring packets to be greater than the number of transport service data units.

Table II Measured delay values for the prototype

	Total	Application (Server)	Transport Service	TR Driver	TR Driver (no copy)
Number of data points	423	424	423	913	913
Maximum value [ms]	567.086	111.542	41.427	15.150	14.170
Minimum value [ms]	20.473	2.110	7.320	2.099	1.068
Mean [ms]	102.995	18.223	19.674	7.122	3.325
Median [ms]	52.037	13.233	18.617	8.529	3.003
Standard deviation [ms]	98.093	14.819	3.510	1.962	0.888

The total delay of the Token-Ring driver includes the copying of a packet to the adapter at the sender side, the transmission of the packet, and the copy operation from the adapter at the receiver. Without considering the copying to/from the adapter, the mean value and the median are 3.3 ms and 3.0 ms, respectively. Knowing that the actual ring transmission time is about 2 ms (assume maximal-sized PDUs of 4300 bytes at 16 Mbit/s), the pure processing overhead of the sending and receiving adapters is about 1 ms. This correlates with the measured minimum delay value (1.068 ms), which can be assumed to be for a very small PDU where the copying and actual ring transmission time is negligible, and the time represents the processing overhead in the drivers and adapters.

The resulting delay, when the delay of the sending-side application and the end-to-end transport delay is subtracted from the total delay, indicates that frames spend the most of the time in the transport service queue at the receiver side waiting to be supplied to the AVK.

Figure 6 shows the **number of frames in this transport service queue at the receiver side**. It is a gantt chart showing the number of frames contained in the queue against the absolute time from the beginning of the measurement. It is obvious that the behaviour pattern of the queue is directly correlated to the pattern of the total delay in Figure 5.

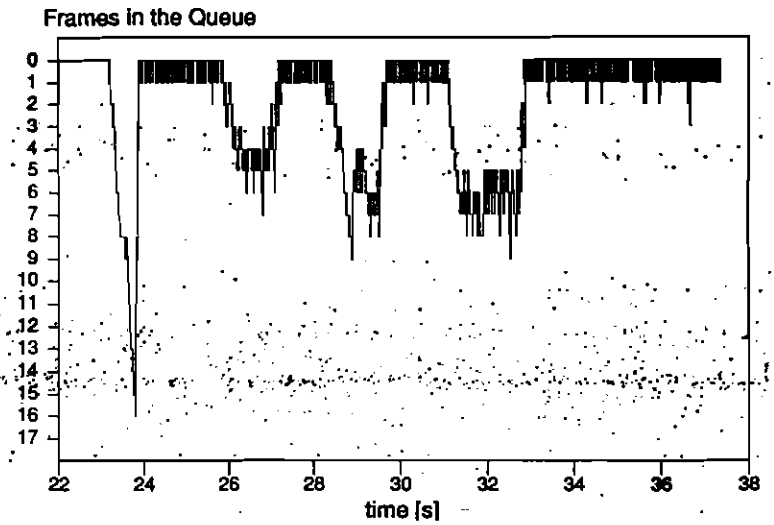

Figure 6 Number of frames in the transport service queue to the application at the client side

It can be seen that 16 frames are queued initially and then supplied to the adapter. This can be explained by the initialization of the delivery card that can only be done after receiving the first frame. In the time that they are displayed, the transport service continues to deliver more frames in a regular pattern. Thus the adapter is always lagging these 16 frames behind the transport service. Figure 4 shows the mean frame size initially at about 5 Kbytes, allowing these 16 frames to fit into the internal bufferspace on the adapter, configured to 128 Kbytes. After transition of a constant screen, the packet size increases, causing the total amount buffered to exceed 128 Kbytes (at least 16 frames at a larger frame size), resulting in frames been queued for delivery to the AVK. The overall delay from the capture camera to the client screen does not really increase but we can only now recognize it in our application, because of the limited buffer space on the adapter.

The total delay from the capture camera to the client screen could not be measured exactly. The delay for a frame given to the card until it is displayed on the screen is missing. All other delay values have been measured. Adding up the individual delay components gives a total

delay of more than 20 frames (660 ms). There are about 4 frames on the capture card, at least 1 frame in the application and communication system, and at least 16 on the delivery card.

It is interesting to see that the transport service does not increase the total delay significantly. The initialization of the delivery card at the beginning of the frame transmission results in a delay value that cannot be decreased during the total life-time of the connection in our system. The system has to be extended with regard to a mechanism that drops frames on the receiving side. It is easier to drop them at the receiver side because only here all information concerning the total delay of the system is available. Since multiple clients can be connected to a single video stream, it is obvious that the server should not decrease his data rate by skipping frames, otherwise some other clients would experience video stuttering.

The delay in this prototype is definitely too high for video conferencing, where end-to-end delay should be less than 250 ms [Heh90]. In our case it is larger. Thus, we need to introduce a mechanism to decrease the delay. One solution is to decrease the buffer space on the delivery card so that only a few frames can be buffered. The stream handler will then have knowledge about additional frames waiting to be displayed. Therefore, it knows the delay incorporated in the system at any time and can decrease it, preferably by dropping frames (*e.g.*, deltas up to the next still frame). Note that such an approach will introduce perceptible audio cracks even though the loss of a video frame is hardly noticeable [Cou91; Jef91]. To reduce the delay, it is necessary to drop both audio and video data. Dropping only one would cause synchronization problems.

Conclusion

The prototype demonstrated at CeBIT'92 shows the feasibility of integrating audio/video into existing desktop computers and networks, as well as the utility of doing so, as demonstrated by the two practical applications that were developed to fruition. One important aspect of both scenarios is the effective mass distribution of high-volume digital data through computer networks. Here the use of a multicast technique with the required protocol changes was used to achieve the goal of distributing data to workstations needing them without overloading other workstations on the network with irrelevant data.

Besides the solution detailed here for OS/2, similar multimedia support components for the RS/6000 under AIX are also being worked on at the ENC. Future work include the development of more powerful protocols to help with multicasting over different types of networks and the integration of AIX and OS/2 systems into one multimedia application.

Acknowledgements

We gratefully acknowledge the work of the whole HeiProject team in conceiving and implementing with us the prototype: Dietmar Hehmann, Ralf Guido Herrtwich, Norbert Luttenberger, Christiane Jungius, Frank Markgraf, Stefan Mengler, Thomas Meyer, Kurt

Reinhardt, Peter Sander, Jochen Sandvoss, Thomas Schütt, Werner Schulz, and Heiner Stüttgen. Special thanks to Kaliappa Ravindran for his valuable comments.

References

[And91] Anderson, D.P. & Chan, P.; **Toolkit Support for Multiuser Audio/Video Applications**; Second International Workshop for Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany; Nov. 18-19, 1991.

[Cra92] Cramer, A., *et. al.*; **The Heidelberg Multimedia Communication System: Multicast, Rate Enforcement and Performance on Single User Workstations**; IBM ENC Technical Report No 43.9212, 1992.

[Cou91] Coulson, G., García, F., Hutchison, D. & Shepherd, D.; **Protocol Support for Distributed Multimedia Applications**; Second International Workshop for Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany; Nov. 18-19, 1991.

[Dee88] Deering, S.E.; **Multicast routing in internetworks and extended LANs**; Proceeding of ACM SIGCOMM 88; Stanford, California; USA; August 1988.

[Dee90] Deering, S.E. & Cheriton, D.R.; **Multicast Routing in Datagram Internetworks and Extended LANs**; ACM Transactions on Computer Systems, Vol 8, No 2; May 1990.

[Gre92] Green, J.L.; **The Evolution of the DVI System Software**; Communications of the ACM; Vol 35, No 1, Jan. 1992.

[Heh90] Hehmann, D., Salmony, M. & Stüttgen, H.; **Transport services for multimedia applications on broadband networks**; Computer Communications, Vol. 13, No. 4, May 1990.

[Her92] Herrtwich, R.G.; **An Architecture for Multimedia Data Stream Handling and Its Implementation for Multimedia Transport Service Interfaces**; Proceedings of the 3rd International Workshop on Future Trends of Distributed Computing Systems, Taipei, Taiwan, 14-16. April, 1992.

[Hut89] Hutchinson, N.C., Mishra, S., Peterson, L.L. & Thomas, V.T.; **Tools for Implementing Network Protocols**; Software - Practice and Experience; Vol 19 No 9; September 1989.

[Hof92] Hofmann, R., Klar, R., Mohr, B., Quick, A. & Siegle, M.; **Distributed Performance Monitoring: Methods, Tools and Applications**; submitted to IEEE Transactions on Parallel and Distributed Systems.

[Hop90] Hopper, A.; **Pandora - An Experimental System for Multimedia Applications**; Operating Systems Review, Jan. 1990.

[Hop91] Hopper, A.; **Design and Use of High-Speed Networks in Multimedia Applications**; 3rd IFIP WG 6.4 Conference on High Speed Networking, Berlin, Germany; March 18-22, 1991.

[Hug89a] Hughes, L.; **Multicast Response Handling Taxonomy**; Computer Communications, Vol 12 No 1, Feb. 1989.

[Hug89b] Hughes, L.; **Gateway Design for Internet Multicast Communications**; Computing Communications, Vol 12 No 3; June 1989.

[Jak91] Jakobs & Quernheim; **Multicast communications in networks with arbitrary topology**; Third IEE Conference on Telecommunications; Edenburg, Scotland; 1991.

[Jef91] Jeffray, K., Stone, D.L. & Smith, F.D.; **Kernel Support for Live Digital Audio and Video**; Second International Workshop for Network and Operating System Support for Digital Audio and Video; Heidelberg, Germany; Nov. 18-19, 1991.

[Kal90] Kalmanek, C.R., Kanakia, H. & Keshav, S.; **Rate Controlled Servers for Very High-Speed Networks**; Proceedings of IEEE GLOBECOM '90, San Diego, USA; Dec. 2-5, 1990.

[Nag92] Nagarajan, R. & Vogt, C.; **Guaranteed-Performance Transport of Multimedia Traffic over the Token-Ring**; IBM ENC Technical Report No 43.9201, 1992.

[Ngo91] Ngoh; **Multicast Support for Group Communications**; Computer Networks and ISDN Systems No 22; 1991.

[Rav92] Ravindran, K., Sankhla, M. & Gupta, P.; **Multicast Models and Routing Algorithms for High Speed Multi-service Networks**; IEEE International Conference on Distributed Computing Systems, Yokohama (Japan), June 1992.

[Sch92] Schütt, T. & Farber, M.; **The Heidelberg High Speed Transport System: First Performance Results**; 3rd International IFIP WG6.1/6.4 Workshop on Protocols for High Speed Networks, Stockholm, Sweden; May 13-15, 1992.

[Sid89] Sidi, Liu, Cidon & Gopal; **Congestion Control Through Input Rate Regulation**; Proceedings of IEEE GLOBECOM '89, Dallas, USA; 1989.