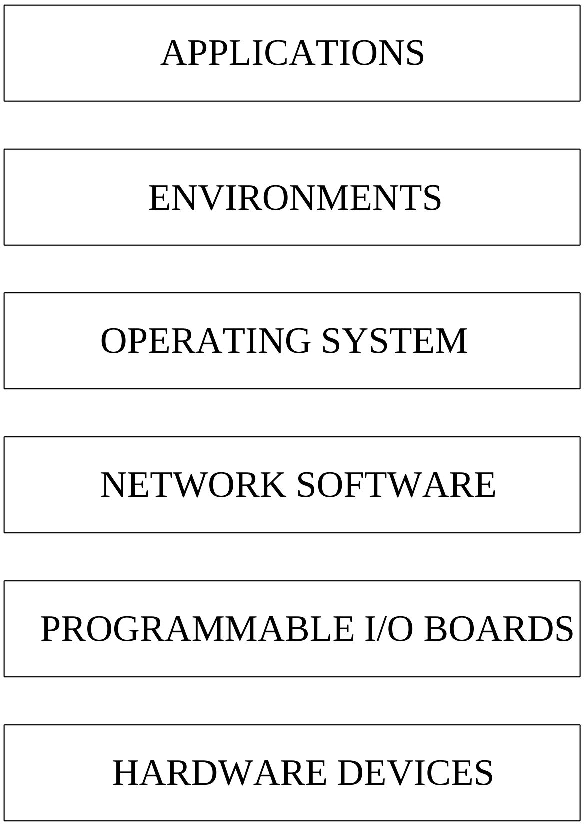

Figure 1: Layers of system abstraction

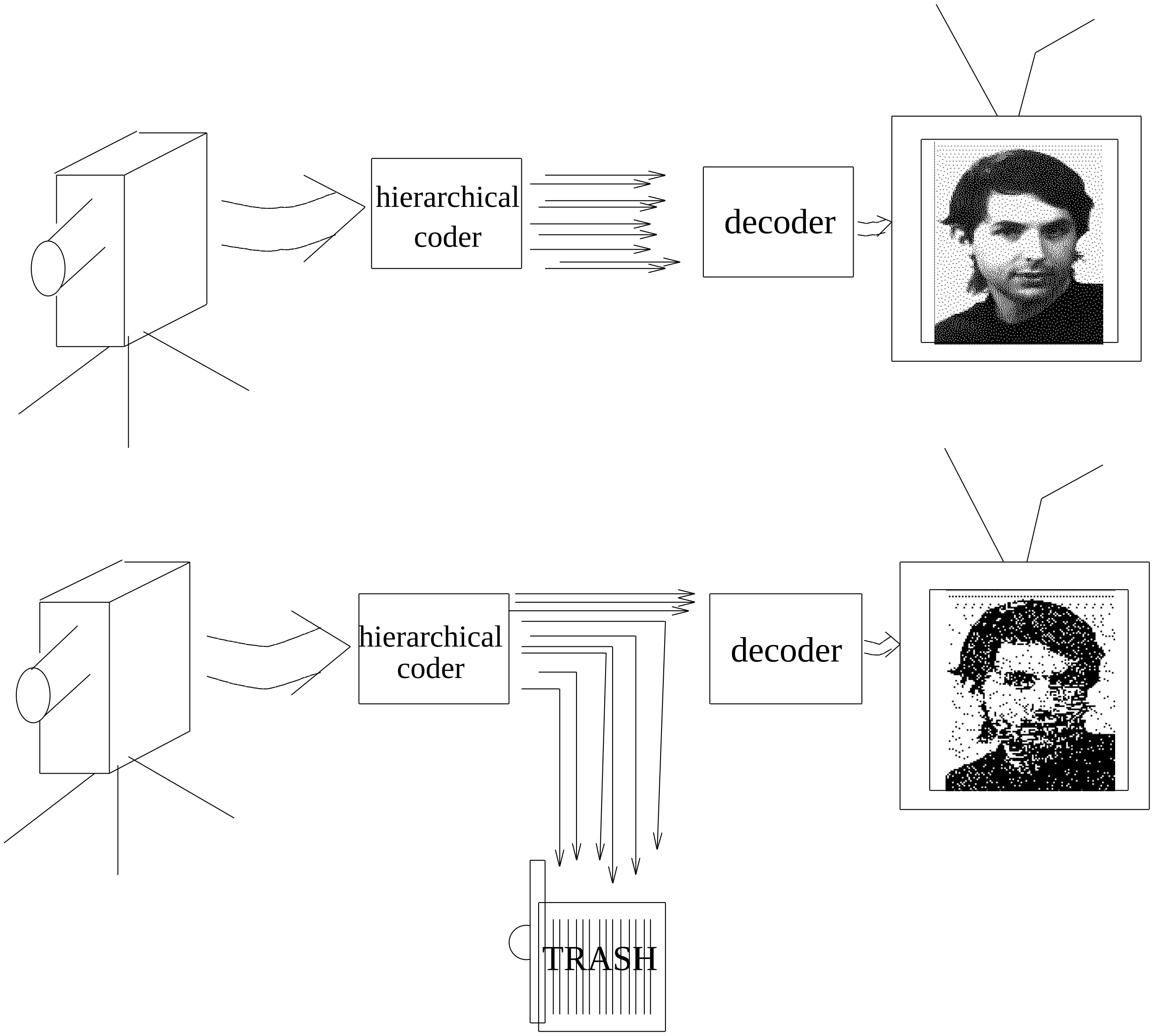

Figure 2: Hierarchical coding and effects of packet loss

Application	Teleconference, etc.
Presentation	Encoding/Compression Error Concealement/Recovery Ciphering
Session	Session Management Quality Tuning
Transport	Error Control
Network Data Link Physical	Communication Subnet Providing QOS ‘‘Guarantees’’

Figure 3: Positioning of Some Important Functions for Multimedia Communications Using Digital Continuous Media with Respect to the ISO OSI Reference Model.

Network and Operating System Support for Multimedia Applications

Joseph C. Pasquale, George C. Polyzos,
Eric W. Anderson, Kevin R. Fall, Jonathan S. Kay,
Vachaspathi P. Kompella, Scott R. McMullan,
and Dipti Ranganathan

Computer Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

Technical Report CS91-186

March 1991

Abstract

This paper describes the general problems in providing system software support for multimedia applications. We specifically focus on applications which have *digital continuous media* (DCM) I/O components, and the type of support that must be provided by the network software and operating system.

DCM applications have large bandwidth and low delay requirements. We discuss network software support to meet these requirements, such as resource reservation schemes based on parameterized requests for some level of quality of service. Most likely, DCM communications will be long-lived, which suggests that connections are the appropriate network construct for such communications.

Regarding the operating system, we describe issues of real-time resource scheduling, synchronization of processes and temporally related DCM flows, shared memory, and the need for a modular, uniform I/O abstraction. Finally, we present design principles based on the concepts of moving processes to data and separating design from implementation as important considerations for system software design.

This research is supported in part by the National Science Foundation, the Powell Foundation, Digital Equipment Corporation, NCR Corporation, and the UC Micro Program. Joseph Pasquale may be contacted by e-mail at *pasquale@ucsd.edu* or by telephone at *(619) 534-2673.* George Polyzos may be contacted by e-mail at *polyzos@cs.ucsd.edu* or by telephone at *(619) 534-3508.*

1. Introduction

In the near future, distributed computer systems will allow users to communicate and collaborate through multimedia applications. Such applications will manipulate and present audio and video data, as well as traditional text and graphics, in an integrated manner.

1.1. Motivation for Audio and Video Communication

Audio and video are important forms of media for human-computer interaction. These media have the ability to convey information which cannot be easily expressed in text. They can also more quickly convey many kinds of information traditionally represented by text. Additionally, receiving information in the forms of audio and video allows us to take greater advantage of our natural sensory abilities.

Indeed, one may argue that the reluctance of the general public to accept computerization is due to the difficulties of interacting through textual commands and output. Graphics are successfully employed to improve user friendliness in many systems, but audio and video will make human-computer interaction fundamentally more natural. New modes of input will be in the form of voice commands or hand and body gestures. Output in the form of sound and video, each of which can be spatially located using a complex of loudspeakers and displays, promises a quantum leap forward over still graphics and text.

Consumer-oriented audio and video products are rapidly becoming computers with specialized hardware interfaces. The programming interface of a modern VCR is not unlike that of an old panel-programmable minicomputer. As these products become more complex their functions will increasingly overlap. This leads to the emergence of devices such as unified remote controls. As computers gain media capabilities, and appliances gain computing capabilities, a common ground is approached. This common ground is the multimedia workstation. This paper discusses the operating system and network support which this workstation will require.

1.2. Current State of Multimedia Systems

There has been a tremendous response from computer, communications, and electronics communities to provide users with multimedia systems. Better hardware technology is rapidly becoming available at reasonable prices. We are witnessing many advances in digital systems, including digital signal and array processors for audio and video, optical storage devices, and fast optical fiber communication networks. Chip sets have become available for important specialized functions such as compression (along with the emergence of corresponding compression standards such as JPEG for still video and MPEG for motion video), and programmable VLSI digital signal processors (DSPs), which have permeated the consumer audio industry. Already, a number of audio and video devices for users owning personal computers have become available.

Higher levels of integration become possible when these digital systems are placed on boards with programmable general purpose processors and memory, such as advanced network interface and graphics boards. Digital Video Interactive (DVI) technology is a good example of a digital video and audio board set capable of accepting high-level commands to perform complex video and audio functions, including on-board compression. Many of these functions are implemented in software, and can be easily modified and improved over time.

We expect the design of new personal computers and workstations to incorporate digital technology in the form of standard digital communications interfaces for applications such as FAX and High Definition Television (HDTV), as well as telecommunications services such as the Integrated Services Digital Network (ISDN) and Broadband ISDN [1]. Future directions in distributed systems development will require the support and integration of multimedia information (e.g., video, audio, dataglove input), to make adequate use of these emerging technologies and to provide a rich framework for creative expression.

Concurrent with the development of new and interesting multimedia devices has been the development of multimedia user applications. Examples include electronic mail incorporating text, graphics, and voice-quality audio, audio-video authoring, hypermedia (i.e., hypertext extended to include multiple media), audio-video teleconferencing, and real-time visualization of scientific applications (e.g., image and volume rendering based on simulations of natural phenomena).

Programming environments and window systems are being developed or extended to include multimedia information. Considerable research is focused on object-oriented programming systems, where the objects may include audio and video information. Window systems are being extended to include video and audio [2].

The requirements of multimedia systems include real-time processing requirements which have to date been largely considered only in the design of embedded real-time systems. However, these systems typically do not have to support a general computing environment. Many of the common distributed systems mechanisms used today (e.g., client/server control, unparameterized communications channels) do not adequately support real-time interactions. Any new mechanisms must accommodate many types of data communication, including both real-time and non-real-time, and make efficient use of available I/O resources while avoiding high congestion or excessive loading.

Figure 1 shows various levels of system abstraction between the user application and hardware devices. There is a great deal of research and development activity at the lower levels comprising hardware, and upper levels comprising applications, and to a lesser degree, programming and window environments. However, much investigation is needed at the middle levels, which we define as ‘‘system software support,’’ encompassing the operating system and network software. (This is not to say that further investigation is not needed in the other areas, but just that the research activity has been greater.)

Interestingly, the current situation of multimedia system design is similar to the early days of computing, when programmers coding applications had to take direct control over devices. Operating systems did not offer sufficiently powerful I/O abstractions for all applications. Similarly, today’s multimedia programmers must write applications which take control over logical devices (perhaps as defined by an objected-oriented programming system, and coordinated by a window system) for a number of reasons. One reason is the failure of most systems to provide convenient abstractions for time-sensitive data. Popular operating system and network software, such as Unix and TCP/IP, do not provide the appropriate abstractions for multimedia communication (e.g., synchronization of multiple I/O streams and real-time delivery guarantees).

Another reason is the inability of existing system software to manipulate data quickly enough to meet the demands of the high-bandwidth low-delay requirements of multimedia I/O. Real-time delivery is required for several types of applications. For example, creating an illusion of continuity for analog output (e.g., audio and motion video), or simultaneity of information capture and presentation (e.g., for teleconferencing or polling remote sensors) requires a bounded delivery delay to be enforced. Consequently, writing multimedia applications requires a great deal of knowledge of multimedia devices on the part of the programmer, which may result in incorrect or suboptimal use of resources.

In this paper we will focus on the problems of providing network and operating system support. The rest of the paper is organized as follows. Section 2 describes the characteristics of digital continuous media. Section 3 discusses issues of network communication. Section 4 discusses operating system mechanisms. Section 5 describes system design issues. Section 6 presents conclusions.

2. Digital Continuous Media

Digital Continuous Media (DCM)[1] refers to digital audio, video, and other digital media, in which data is produced or consumed in a periodic manner. DCM sources generate data at regular intervals related to the sampling frequency of the analog input. For example, a video frame is typically generated every 1/30-th (or 1/25-th) of a second. Coding and compression of the information generated at these instants may result in a variable bit rate (VBR), which depends on the actual information content of the signal and allows for efficient, constant quality communication. We note that continuity is a property of how the media emanates from the source, and how it is ultimately presented to the user; however, it need not characterize how the media flows, e.g., how it is transported over the network.

Audio and video are often associated with real-time applications such as teleconferencing. The interactive nature of this application can impose stringent delay requirements on the audio and video flows. If the time required for this data to travel from the sender to the receiver cannot be kept within a certain bound, the data cannot be effectively used for interactive communication. Various guidelines set this tolerance to tens or a few hundreds of milliseconds [3]. Keeping the maximum delay within bounds is not sufficient for continuous communication. Buffering limitations can impose the additional constraint that the maximum inter-arrival time of packets at the destination must also be bounded.

Meeting delay requirements may require dropping packets when they accumulate between the sender and the receiver. Packets can normally be processed only at a certain rate, so if accumulated packets are not discarded, the receiver cannot catch up unless the source stops sending packets. Fortunately, humans receiving real-time audio and video have some tolerance to transmission errors. For example, for telephone quality audio, a 1-2% data loss

[1] The term ‘‘continuous media’’ is used to distinguish information which is continuous in nature, such as audio and motion video, from the more general types of information encompassed by the term ‘‘multimedia.’’ [2]

during talkspurts is usually acceptable [3]. The perceived error may vary, however, depending on the particular encoding schemes used and the level of compression. These two observations lead to a characterization of DCM as *soft real-time traffic*. This traffic requires specific performance levels for acceptable behavior, but has some tolerance to data loss when the requirements cannot be met [3].

Another important characteristic of DCM is their very high traffic volume. For example, telephone quality voice requires a digital data rate of 64 Kb/s. For CD quality stereo sound, bandwidths of 1.4 Mb/s will be needed. Digitization of a video frame measuring 640×480 pixels requires about 7 Mb. Motion video, transmitted at 30 frames per second, requires about 220 Mb/s of bandwidth. These requirements can be significantly reduced with efficient coding schemes and compression techniques. Frequently, the degree of compression applicable to DCM can be augmented by applying lossy techniques which suppress less critical characteristics. For most broadcast quality video, 30-45 Mb/s is sufficient. Throughput requirements such as these are beyond the capabilities of most current systems.

New hierarchical coding techniques split DCM signals into subcomponents of varying importance [4]. These methods are also referred to as layered, subband, or component coding. Less important components can be dropped without causing an interruption of service. Communications can continue in the presence of delays, errors, or network congestion, with only a drop in service quality. The simplest form of component coding decomposes samples (e.g., video frames) into a prioritized set of sub-samples. This is important, as it gives the system software the capability of deciding how to gracefully degrade transmission dynamically when resources are limited. Figure 2 illustrates two cases of video frame reconstruction, one where all sub-samples are used, and one where only a few of the highest priority sub-samples are used. The most effective coding and compression techniques will be sophisticated hierarchical methods, based on band-pass filtering in three dimensions (2D image and time) [4].

3. Network Communication Issues

Several key issues relate to communication channel implementation. Most of these issues are traditional points of controversy, which have been resolved in some sense for conventional data communications. However, the demanding performance requirements of DCM, and the inadequacy of current systems to support these requirements, motivate us to reconsider these issues, from the perspective of DCM.

We first define some terms which we will use in this paper. A *flow* is a time ordered sequence of DCM samples. A flow originates at a single endpoint, the sender or transmitter, and terminates at one or more endpoints (receivers). For example, the output of a video camera after digitization would be a flow, typically composed of frames which are generated periodically. When component coding is used, each of the components of the video signal could be considered a distinct flow. Flows are transported by *channels*, which are generally packet-switched virtual circuits or datagrams, but which could be circuit-switched channels. *Isochronous* communication refers to delivery of information within some bounded time interval, or the delivery of a sequence of samples at regularly spaced intervals of time, usually over a network. Support for isochronous communication is needed to create the illusion of simultaneity between a process relaying captured DCM samples to its associated receiver via an interposing network.

It is instructive to attempt to position the various functions, transformations, and services performed in relation to DCM with respect to the ISO OSI reference model [4, 5]. Figure 3 shows the most important functions and their suggested layering. Notice that there is no *a priori* set boundary between user and system space, between operating system and network software, or between hardware and software. We will focus mainly on functions at the presentation, session, transport, and network layers.

3.1. Longevity of Communication

DCM communication is inherently long-lived, particularly when compared to traditional data interactions. For example, it is expected that teleconference sessions will typically last for at least many minutes. Furthermore, there is significant predictability of the generated traffic. In addition, DCM traffic requires delivery, or at least presentation, of samples in order of generation. All these points suggest that connection-oriented communication models and services are the most appropriate for such applications.[2]

[2] A form of fast packet switching, called the Asynchronous Transfer Mode (ATM), has been adopted as the preferred switching technique for Broadband ISDN. ATM is based on virtual-circuits, i.e., it is connection oriented, and uses short virtual-circuit identifiers to reduce overhead to the small (fixed) packet size (which has been selected in order to facilitate audio communication by avoiding long packetization delays, and thus, echo problems).

Connectionless communication involves some additional overhead for control information. Data is transferred in packets, each independently of the others. As a result, each packet must include enough information to carry it to the destination, without regard to how the other packets get there. Furthermore, since packets are not guaranteed to arrive in any specific order, sequencing of packets may have to be provided by a higher protocol layer. Because multimedia objects are large, and have short lifetimes when destined for real-time presentation, buffering and resequencing is difficult. Furthermore, the demands placed by multimedia communication necessitate some guarantees by the system that it would have the resources available for data transfer and processing. In a connectionless model, this is difficult to achieve because of the lack of state information.

On the other hand, connection set-up cost is an important factor in the success of connection-based systems. If it is expensive to set up and maintain a connection, sporadic communication may be better supported by a connectionless mechanism. For example, in a tele-lecture, there will be a connection from the speaker to all the listeners. The listeners will probably have no connections among themselves because the cost of maintaining all these connections will be prohibitive, particularly because they may never be used. However, if someone in the audience asks a question, a connectionless mechanism may be the appropriate way to distribute the question to all participants, in view of the short-livedness of the communication.

3.2. Connection topologies

We foresee multimedia applications requiring one-to-many and many-to-many connectivity for multiple DCM flows and multiple participants. When communication using audio and video is supported, we believe that human interaction through the system will be natural. This would motivate users to have group interactions over the network. A single connectivity model should be able to capture these group interaction topologies, as well as the more conventional one-to-one and one-to-all communication. To support these connection topologies, Leung has suggested a new operating system object, called a ‘‘connector,’’ which is a switchboard connecting multiple inputs to multiple outputs [6].

An important property of connections is reconfigurability. During the course of a session, a connection could be reconfigured for several reasons. First, the connectivity of the session may change, with participants either leaving or joining the session. Second, the resource availability may change because of system failures or congestion (e.g., too much traffic on the network, not enough memory available in the system, etc.).

3.3. Efficient Data Movement

Due to the high data volume of DCM it is important to provide efficient data transfer mechanisms even when high-speed fiber-optic networks are available.[3] Therefore, support for efficient multicasting is an important requirement, well recognized [7], but poorly supported at the present time. *Multicasting* refers to one-to-many communication, in contrast to *unicasting* which is one-to-one communication, and *broadcasting* which is one-to-all communication [8]. Multicast can be thought of as intermediate service between unicast and broadcast, and, unfortunately, many times it is implemented as either a broadcast (with filtering at higher protocol layers), or as a series of unicasts from the source.

The naive approaches to multicasting described above are inefficient because the network may be unduly overloaded. Broadcasting sends the data to many more nodes than required. Multiple unicasts could use the same link several times to carry the same data, and the sender must work harder to send the multiple copies. However, multicasting can be done efficiently using tree-based communication structures [9]. These structures can reduce bandwidth utilization by reducing the number of copies of data sent over the network. This is achieved by delaying duplication of data until the routes to different destinations diverge.

In addition to reducing bandwidth requirements for multimedia communication, routes that have low end-to-end delay from the source to possibly multiple destinations must be found. We would like to have data paths that avoid crossing too many network relay nodes and many abstraction layers (e.g., protocol layers) because they introduce delay in data delivery. This delay may be due to switching, queuing, protocol conversion, etc. Propagation delay must also be added to the equation. However, taking into consideration the fact that delay is tolerable until it exceeds some threshold level, a good routing algorithm may not have to minimize the delay along the route; bounding the delay should prove adequate.

[3] History has shown that there is no such thing as an abundant resource. Ethernet is a good example of this.

An interesting interaction between component coding and multicast communication is the following. Component coding gives destinations the facility to adjust the quality of the signal they receive independently of each other, without the source necessarily being aware of it. Thus, if they cannot obtain the required resources for high-quality reception, they can opt for a lower resolution, by accepting packets from only the low-quality component flows. Of course, the adjustment is possible only towards lower quality, but the source could generate a high quality signal.

Multicasting has been mainly considered to be a networking construct. However, it also embodies concepts which are equally applicable in the operating system. The nature of multimedia data emphasizes the need for a resource-conserving transport mechanism, whether over the network, or within the confines of a single machine. On the local machine, efficient data transfer with delayed duplication translates to using shared memory, as described in Section 4.4.

3.4. Performance Guarantees

In non real-time systems, services are provided with a *best-effort* approach; the correctness of an application is independent of actual system performance. However, for general purpose distributed systems to be able to handle traditional *time insensitive* workloads as well as new *time sensitive* DCM, careful system design and resource allocation are required. Since DCM applications typically require a certain minimum set of resources to be available in a timely manner in order to behave correctly, it is of utmost importance to be able to express resource needs in a manner that both minimizes the potential for degradation of service to the application and maximizes system productivity.

Performance requirements can take many different forms, some of which are easier to express, provide, or verify than others. Deterministic performance requirements are the easiest to express and understand, but the most difficult to satisfy. They are given as upper or lower bounds on performance metrics such as transfer delay and data rate, and are also usually referred to as *hard real-time.* The problem with deterministic guarantees is that it is impossible to provide them in the presence of the possibility of errors. Therefore, statistical guarantees have been suggested as a more economical and usually adequate alternative. These are usually specified in the form of a bound on the probability that a given performance metric is within a given range. A different problem arises with statistical guarantees, as defined above: there is no time frame specified for checking whether the advertised performance targets have been met or not. To overcome this difficulty, ''fractional'' guarantees have been suggested, where a specific horizon is introduced. For example, one could require that *n* of the last *m* messages satisfy the given performance criterion[4] [3].

3.5. Quality of Service Specifications

A quality of service (QOS) specification should be able to fully express an application's service needs while providing information necessary for the system to meet those needs. It has been suggested that the application itself is best equipped to describe its own requirements [10, 3, 11, 5]. However, the parameters that succinctly capture these requirements may not always be entirely clear. There have been a number of efforts to determine both the necessary QOS parameters and a complementary resource reservation algorithm. Values for maximum message size, data rate, and burst rate, along with a proposed reservation protocol defined as an extension to Internet Protocol (IP), have been described by the DASH system [10]. Little and Ghafoor define QOS as a six-tuple consisting of a speed ratio (ratio of actual to nominal data presentation rate), utilization ratio (ratio of data presented to data available), average delay, maximum jitter, maximum bit error rate and maximum packet error rate [11]. Ferrari characterizes a set of guaranteed real-time communication channels with the parameters of channel capacity, maximum packet size, per packet delay bound, and maximum packet loss rate. These parameters are used in the specification of a scheduling algorithm guaranteeing performance in a packet-switched wide-area network, utilizing processing at intermediate network nodes [12]. Regardless of the parameters chosen, they should be rich enough to facilitate scheduling, synchronization, buffer allocation, and flow control [10, 11, 5].

At some point during a connection's lifetime, QOS parameter refinement may become necessary in the face of events such as dynamic changes in multicast group membership or requests for improved QOS, e.g., changing video-image resolution. One way to accommodate this situation is to require both the reservation protocol and the scheduling algorithms to be *adaptive*, allowing the connection to remain in effect without the need to re-establish an entirely new connection.

[4] An apparent difficulty that arises here is that if any node other than the source attempts to verify the performance guarantee, it would be unsure of whether it has received all the last *m* messages. This can be resolved by requiring a traditional sequencing mechanism.

A system might offer a service which provides various distinct, well defined connection classes satisfying requirements for typical applications, thus greatly simplifying DCM application coding. In addition, long-lived or frequently accessed connections can perhaps persist in the system in an effort to reduce setup times for commonly used applications utilizing such connections.

Most current packet-switched networks and operating systems do not support QOS specifications for multimedia communication. A major effort is currently underway to specify and implement a network layer protocol, ST-II [13], which supports many of the desired services for DCM, such as flow connections with QOS parameters, efficient multicasting, etc. To achieve these objectives, unlike the Internet Protocol (IP), ST-II keeps connection state information in gateways.

3.6. Resource Reservation

For performance guarantees to be provided, an appropriate level of resource reservations must be made. Note, however, that in the case that the guarantees provided are not deterministic, the reservations required can also be statistical in nature. For example, given several flows traversing a node, traffic statistics for these flows can be used to determine the level of resource usage that will be required. Then, an allocation of resources can be made which satisfies the given performance tolerance probabilistically.

A common unit of allocation in network communication is a *connection* (virtual or physical), between a source and at least one sink. A set of QOS parameters are associated with this connection, and appropriate resource reservation (if any) is attempted in an effort to enforce the QOS specification. Such reservations are appropriate (i.e., efficient) when the connection setup time is much shorter than the service period. Note, however, that reservations could be time multiplexed to fit the periodic data generation patterns of DCM. For example, if isochronous transport is provided, resources could be reserved for a particular flow only for an interval close to the expected arrival time of a sample on this particular flow, and could be reserved for other flows at other points in time. Furthermore, resources can be released early if service for a particular sample is shorter than the reserved interval, or if the sample has missed the deadline (which is important because these situations will usually arise during congestion periods, when system resources are particularly valuable).

To handle the execution of applications with different time sensitivities in a general purpose distributed system, the scheduler must be able to deal with the spectrum of performance requirements from hard real-time through statistical guarantees to traditional best-effort requirements. Deadline scheduling algorithms have been proposed and should be applied in order to achieve the required performance goals [10, 12]. A less exacting alternative without deadline scheduling, based on a combination of priority based mechanisms and admission control for flows, can use a conservative approach for allowing flows to be accepted at higher priority levels (only whenever this does not violate the requirements of other flows at equal or higher priority levels), and monitor individual flows to make sure they do not violate their QOS contract.

3.7. Routing

The routing of DCM data will require not only point-to-point connectivity information which is used by current routing algorithms, but also policy or QOS information in order to construct routes that can support the required level of service. For group applications, the routing protocol of choice must provide routing for point-to-point source-destination pairs, and also for a potentially large number of participants in a multicast group.

Source routing, in which the source specifies all intermediate nodes, is one possible strategy for route determination in high-speed networks because the source does all the computation and requires minimal processing at intermediate nodes. For example, a connection-based multicast route using trees could be computed by the source of the multimedia data. In order to achieve this, it may be necessary for the source to negotiate with relaying nodes to ascertain that resources will be available to carry out the communication. Thus, resource reservation may be a part of the connection setup phase. However, source routing may not be an acceptable solution for large dynamic networks where the state of the network changes frequently over time.

3.8. Work Distribution

Connections will have some quality of service specified, and the network will have to be able to make guarantees when possible. An important question that remains is where to put the processing needed to provide such guarantees. One contention is that beyond resource reservation and routing information, little must be demanded of relay nodes. The endpoints should do all the processing required to present data to the application with the specified quality. For example, with high-speed low error-rate fiber-optic based networks it is widely accepted that error control and sequencing are functions that intermediate nodes should not be handling. For other functions, the questions are not settled since it can be argued that jitter control, for example, may be best controlled by making sure that each relay node transmits the data very close to its deadline in order to provide as predictable traffic as possible.

The use of light-weight protocols is particularly important for DCM. The functionality of current protocols includes such features as error control through error detection and retransmission, which may not be necessary, or are ineffective for DCM. First of all, fast fiber-optic links have very low bit error rates (of the order of 10^{-12}). Second, human tolerance to errors in audio and video is remarkably high. Most importantly, because of the propagation delay which cannot be eliminated even with fast networks, a retransmission-based mechanism might be useless because retransmitted packets will probably violate the delay requirements of the application, and thus cannot be used.

Timer management for error control through retransmission, and checksumming for error detection are time consuming operations in current network protocols [14, 15]. If these services are not required, or are unusable by DCM applications, there should be mechanisms to bypass them. If possible, light-weight protocols which provide only the absolutely necessary services, should be adopted. This approach has also the potential to address the duplication of services among protocols and protocol layers, a very common problem with most current protocol architectures.

An extreme case of this concept is virtual cut-through, where the functions provided at intermediate nodes are minimal and the forwarding of the packet is initiated before the whole packet is received, which minimizes the delay introduced by the relay. This is particularly important for large packets, but less critical for networks which adopt small packet sizes such as the ATM based B-ISDN.

4. Operating System Mechanisms

Operating systems should provide mechanisms for synchronization of processes and DCM flows, real-time scheduling, and shared memory. We discuss these issues and their relevance to supporting digital continuous media.

4.1. Process Synchronization

As time is so important for DCM communication, synchronization is a key issue. Mechanisms for synchronizing processes with each other (i.e., imposing a time ordering on specific execution points within each process), synchronizing a process with events, particularly points in real-time (i.e., a tick of a real-time clock), and finally, synchronizing DCM streams with each other (i.e., imposing a time ordering on specific frames within each DCM stream), are all important in supporting distributed DCM communication.

The two common modes of communication are *synchronous* and *asynchronous.* Synchronization is the coincidence of two events at some point in time; such a point is called a *synchronization point*. Synchronizing processes causes the specific execution points of two processes to coincide. A process performs an operation in synchronous mode if it blocks until the operation completes. This synchrony guarantees that the operation will complete before the process begins another.

A process can use synchronous inter-process communication (IPC) mechanisms to ensure that some operation in another process is performed at a particular time. An example of a synchronous operation is waiting for another process to send a request. Thus, the time during which the requesting process is blocked can be considered to be a synchronization point, since each process can determine that the other process is at a particular execution point. Likewise, if the process to which the request is sent waits for that request, then that can also be considered a synchronization point. Synchronous IPC is generally needed when control functions accomplished by a number of processes must be strictly ordered, and in the case of DCM communication, when setting up connections and reserving resources.

Synchronous IPC allows processes to logically order their activities relative to each other. However, the precise moment this happens is not generally controllable by the processes. It is yet another issue to synchronize a process with a real-time event (which may be process specified or not). Furthermore, the synchronous execution of two processes constrains parallelism, because either one process or the other is always blocked.

A process performs an operation in asynchronous mode if it does not have to wait for the operation to complete. A process can use IPC to cause some event to occur in another process, but in such a mode, neither process can determine the state of the other process. No synchronization point exists. Messages can be sent asynchronously, allowing the sender to immediately resume without waiting for reception of the message by the intended receiver. If messages are received asynchronously, i.e., the intended receiver is not waiting for the message which arrives unexpectedly, a mechanism can be provided for alerting the process that this has occurred, such as an interrupt or a bit which can be polled.

Asynchrony promotes parallel execution of sending and receiving processes because neither needs to block on the action of the other. Asynchronous IPC is relevant to DCM communication whenever a sender need not be synchronized with a receiver (e.g., multimedia mail), and when communication of a DCM stream is not always regular (e.g., stopping transmission during periods of silence for audio).

Under both synchronous and asynchronous modes, points of real time are unrelated to communications arrival events and synchronization points. However, the progression of real time is fundamental to the isochronous mode of process synchronization. An operation is performed in isochronous mode if the process which causes the operation to occur performs the operation at regular intervals in relation to some scale such as real time. If a process can read the current time, it can determine the number of operations the other processes have performed. Isochronous operation allows pipelined parallel execution of processes.

One way isochronous operation can be achieved is if the kernel provides processes with an alarm clock mechanism, which can be set to generate an event (such as the reception of a software interrupt or a kernel-generated message) to occur at a future point in time. A process can then wait for this event. (After it occurs and the process is awakened, it must actually obtain physical processing and memory resources to actually execute. This is provided by resource, e.g., CPU and memory, scheduler(s), which may or may not be integrated with the synchronization mechanisms.) Once executing, the process can reset the alarm. By continually repeating this sequence of steps, a process can generate a sequence of timer events regularly spaced in time, and can then synchronize to them. This is relevant to DCM communication in that the frames of a DCM stream must be captured, or presented, at regularly spaced points in time to simulate continuity.

4.2. Real-time Scheduling

Unlike most textual data, a DCM sample is usually characterized by a presentation deadline after which its utility is greatly diminished. To maintain temporal continuity, the system must transport DCM samples from source to destination and have the destination process react by some deadline to present DCM samples. Mechanisms for real-time resource scheduling attempt to ensure the availability of physical processing resources (e.g., CPU, physical memory) within bounded time.

Unfortunately, resource demand typically exceeds resource availability. A preempting resource scheduler employing priorities provides the functionality to meet real-time resource demands during periods of high contention. This assumes that there are at least enough resources for the high priority processes requiring real-time response, and that low priority running processes can be preempted. Priorities include not only traditional CPU scheduling priority, but may also include real-time deadlines or lag time tolerances. Whenever possible, it is advantageous to allow processes to *work ahead [2,* 16], so that processing is done prior to deadlines if resources are available, to get work out of the way as early as possible.

Scheduling of network resources for DCM flows is also needed. Isochronous communication requires reservation of bandwidth to avoid network buffering. Intermediate node buffering in a network introduces unpredictable delays, complicating the synchronization problem already present at a communication receiver intended to operate in isochronous mode.

During periods of congestion, the network may also employ prioritization and preemption. Hierarchical decomposition applied to a DCM flow, resulting in a set of prioritized flows, can be used advantageously during periods of high congestion. Graceful degradation results from dropping the low priority flows and keeping the high priority flows, which are the most important in reconstructing the original flow.

4.3. Temporal Composition

Applications which need to integrate several types of media require a mechanism to combine DCM flows according to some well-defined temporal ordering. Samples from distinct DCM flows collected at the same time or which must be presented at the same time require inter-flow synchronization primitives to provide temporal composition.

One way to achieve synchronization is for the sender to bundle several flows together on a *single* virtual-circuit using time division multiplexing. Properties associated with virtual circuits [6] (e.g., order preservation and duplicate elimination) are good for achieving smooth interleaving of multiple flows, because the receiver need only route incoming flows to their appropriate destination devices, thereby reducing synchronization complexity at the receiver. Unfortunately, this scheme cannot take advantage of hierarchically coded flows (unless packets can be selectively dropped, which goes against the properties of virtual circuits). Furthermore, source multiplexing from several geographically distributed senders is extremely difficult; having them all send to a single process charged with merging the flows would require such a process to synchronize the frames anyway, which this technique is trying to avoid. Perhaps the most serious disadvantage to this scheme is its inefficient use of communication channels, because the channel must provide a level of service adequate for the most demanding flow.

Another approach to inter-flow synchronization is to allow the flows to be sent over separate communication channels, and have the receiver merge temporally-related frames. When a frame is received, any required spatial composition can take place, such as composing hierarchically coded flows, or mixing of related flows (e.g., two audio flows destined for a single loudspeaker). When a presentation deadline is reached, the receiving process delivers the samples it has successfully composed to their destination devices. This approach has the advantage of routing different DCM flows over different communication channels, each of which may have different qualities of service. By reserving channel resources only to meet the requirements of the flows they carry, channel utilization is maximized. Of course, the synchronization process at the receiver is complicated under this approach, because temporally-related frames do not necessarily arrive in any predefined order. We note that this complexity can be overwhelming. This is in contrast to the simplicity of using virtual circuits which has been used very effectively in existing systems [6].

An interesting inter-flow synchronization scheme worth noting is that of logical time systems (LTS) [2], where a set of related media flows are associated with a common time-frame and clock rate. All timing information for the flows in a LTS is relative to the common start time. Each flow in the LTS terminates at a device. After the LTS is started by a client, a server ensures that all the devices start presenting data simultaneously. To present data continuously, all the devices must have received enough data so that they do not starve, and all the devices must be ready to present the data. Once the LTS is started, its common clock advances monotonically at the set rate. Each of the I/O devices will run at its own particular rate. To synchronize the device with the LTS, the effective rate is adjusted by repeating or skipping frames or samples. The rate of the LTS may then be set at the rate of the device that is most difficult to vary and the other device rates can be ‘‘varied’’ to synchronize with this rate.

The LTS provides a basis for output and input synchronization with the aid of time stamps. Clients can also request events occurring at particular LTS times, for example, pausing at a particular point in a video. An important attribute of an LTS is that clients can schedule and tag data flow with respect to a common start time, thus temporally relating the different media flows.

4.4. Shared Memory and Data Movement

The ability to share memory between processes is especially useful in processing both multiple DCM flows, and individual ones. For instance, spatial composition of DCM flows can benefit from a shared work area where the composition may take place. Examples include mixing audio from separate sources or reconstructing a DCM flow from a set of hierarchically decomposed flows.

In processing an individual DCM flow, it is often the case that it must go through several transformations, accomplished by different modules (e.g., protocol processing). In general, these modules will modify the incoming packetized data, and may also prepend or append data. Movement of data is time-consuming, and if it can be avoided, it should. To avoid movement, if the modules are within the same address space, arrays or linked lists of pointers to logically sequenced (but physically dispersed) blocks can be used to avoid physical movement of data. If the modules are in separate address spaces, dynamic virtual memory mapping can be used, if the data is aligned on page boundaries. I/O system calls should allow the flexibility of having the kernel decide where to place data (e.g., on a page boundary or a specific offset from a page boundary), and return the location to the process, rather than the process specifying a location a priori. The kernel can then insure that I/O data is page aligned according to a process’s specification.

Because DCM is typically accessed sequentially, it can exhibit little or no locality of reference. Given the large size of DCM, it can easily render caches useless by pushing out data which, unlike DCM data, is expected to be referenced again. If this can be determined prior to access, a mechanism which allows caches to be conditionally bypassed would solve this problem. On the other hand, one can take advantage of the expected sequential access of DCM, by prefetching it so that when a process needs it, it is in the cache (e.g., physical memory).

Finally, since DCM flows are continuous (i.e., DCM samples are synchronized with regularly spaced points in real time), the possibility of directly connecting DCM I/O devices, and having DCM be transported directly between a producer device and consumer device, *without user process intervention* is possible. If the kernel provided a mechanism for sharing DMA buffers, a data path between the devices could then be constructed which resided completely within the kernel. This would significantly reduce data copying overheads, which is especially important for high-bandwidth DCM flows. Control of the data transfer could be provided by kernel-level threads operating isochronously.

5. System Design

We now describe some recommendations for system design to support digital continuous media. Many of these recommendations are based on well-established ideas, which we extend to digital continuous media.

5.1. I/O Abstractions

An abstraction for an I/O channel is needed so that all sequential I/O can be done in a uniform fashion. The channel abstraction should be independent of the type of data that is communicated, and should transport digital continuous media. The I/O channel abstraction models communication between a process and a device, and a process and another process (IPC). It is even possible for an I/O channel to form a data path directly between devices; when such a device generates an interrupt one can transport the associated data. The channel abstraction should also be network transparent, so that communication between two entities separated by a network is accomplished in the same manner as between two entities on the same machine.

To advantageously incorporate DCM devices in the channel abstraction, we need a richer interface than the set of UNIX I/O-related data-transport system calls such as *read()* and *write()*. For instance, it would be useful to have *start()* and *stop()* system calls, which respectively activate and deactivate a continuous flow of DCM. *Start* and *stop* are necessary to solve problems resulting from channels which drop data instead of automatically blocking the sender. One problem is that large quantities of data could be dropped and lost if receivers are not prepared for a DCM transmission. Another problem is that the sender is not blocked. Extensive resources could be wasted if the sender starts sending far in advance of when the receiver is expecting the transmission.

The interface should be uniform over all types of devices, and apply to all I/O, not just DCM-related I/O. *Start()* and *stop()* should also be implemented for other types of I/O devices. For example, the effect of *stop()* upon file I/O would be that no data would be allowed to move either toward or from the disk or disk buffers.

Many interesting functions can be provided for I/O channels. The ability to dynamically form connections between two I/O channels, so that data from one is automatically transferred into the other, is useful for data transfer set up by forwarding entities. Also, the operating system may support a variety of *filters* which can be applied to I/O channels. Filters transform the data flowing through an I/O channel. Restricting all but the most important of hierarchically coded DCM flows is an example of a filtering operation.

Filters are similar in concept to the modules which can be pushed onto a stream in AT&T Streams [17]. In addition to their data manipulation functions, filters can act as splitters or joiners (i.e., a filter in reverse). In addition to the AT&T Streams functionality, we envision a facility to allow filters to be propagated toward the source end of an I/O channel. For instance, if a process on the receiving end of an I/O channel places a filter on that channel, and the channel spans across a network, propagating the filter to the transmitting end would allow significant reduction of network resource use, especially for high-bandwidth DCM flows.

5.2. System Structure

There are many ways to structure system software, each of which has its own advantages and disadvantages. One structural issue is the question of how much functionality should be placed in the kernel. For example, an arriving DCM flow is likely to be processed by multiple modules on its way from the network to the receiving process. If each of these modules resides in the kernel, the only time the user-kernel protection boundary is crossed (which can be a costly operation), is during process I/O for user-to-kernel and kernel-to-user copies. Such copying can often be eliminated by using shared memory and dynamic virtual memory remapping techniques. Use of memory remapping can be provided if the kernel is permitted to page-align I/O data.

Instead of using virtual memory remapping to eliminate kernel-to-user data copies, one can simply remove the user-kernel protection boundary altogether. This method will improve performance by removing copying and address space switching for process I/O. Applications can then interface directly to appropriate modules, such as a particular network protocol layer. However, the loss of protection is generally too costly for this technique to be practical.

An approach gaining popularity is to have a small kernel utilizing user-level processes for higher level functions. This structure is flexible, protection is enforced through message passing, and efficiency can be improved by integrating virtual memory and message passing as is done in Mach [18]. Of course, if the cost of context switching is high, this approach can be inefficient.

Finally, one can have a modular I/O system by providing a mechanism similar to AT&T Streams. Stream modules reside in the kernel, and are dynamically configurable by user processes. Much copying between stream modules can be eliminated using shared memory and pointer modification techniques. Protection is enforced, context-switching is limited, yet there is greater flexibility than the kernel structure described first.

5.3. Design Principles

A principle for systems supporting DCM is to *move processes to data* rather than moving data to remotely located processes. For instance, if a video board has a programmable processor, it is advantageous to execute code (e.g., a process) there which would compress a video frame as it was captured and then send the smaller compressed frame to its consumer, rather than to move the large uncompressed frame to a remote processor. This reduces the required bandwidth, and thus increases performance. Another example is the use of a general PostScript printer. The traditional method of graphic output to printers, before the invention of printers with powerful CPUs, was to render the entire image within the host computer and then transfer it. For a PostScript printer, the code is a PostScript program which generates a printed page. The data is the vast amount of vector font data stored within the printer. In this case, moving code rather than data is clearly beneficial. Finally, the idea of propagating filters toward the data sources is yet another application of the principle.

The principle of moving processes to data can be applied to I/O board design. Given that I/O boards of all types (network, graphics, audio) can be constructed to support their own processors and memory, one can place a small operating system on those boards to support processes which can be initiated remotely. Leung et al [6] had such a scheme, where they placed a small real-time kernel on their multimedia workstation's network interface board. Such a general purpose system is not only good for performance, but is very flexible and extensible. This also stresses the importance of having a uniform abstraction for I/O.

A functional design is often layered. Defining progressive levels of abstraction helps control complexity in design. If this layering is maintained in the actual code structure, however, it may not perform well because time-consuming functions are repeated at each layer. To avoid these inefficiencies one can *separate functional structure from the structure of the implementation*. A good example occurs in network protocols, where different layers of protocols perform error detection and correction, such as checksumming, at multiple levels, when it could be done once at a high level. High-bandwidth delay-sensitive data exacerbates such performance problems. This problem is well-known; Clark and Tennenhouse describe an approach to solving this problem for network protocols [19]. They suggest identifying functions performed at multiple levels, and they suggest how to avoid repeating such functions.

6. Conclusion

We have presented the primary issues relevant to supporting DCM applications by the network software and operating system. Real-time guarantees are needed to ensure continuity when presenting DCM. These guarantees must be provided without the system being overly conservative in its allocation of resources. Both transmission delay and physical resource allocation times must be bounded. Mechanisms are needed to synchronize multiple DCM flows with themselves and with real time.

Providing both guaranteed and best effort I/O performance simultaneously to an application requires a uniform I/O model provided by the underlying operating system. Any such model need not only address data produced by discrete media devices (e.g. disks, tape drives), but must also consider the real-time characteristics of DCM. Priority-based synchronization and scheduling algorithms are needed to ensure these guarantees are met.

General connection models should be supported by both the network and the operating system. This would involve constructing efficient routes that satisfy the quality of service the application demands. The models should also be dynamically reconfigurable to adapt to new topologies and changes in quality of service.

Characteristics of DCM flows complicate software design by requiring a large volume of data to be manipulated in a limited amount of real time. Design principles which minimize movement of data (e.g. by moving processes to data and collapsing conceptual processing layers in implementation) are effective for implementing system software offering guaranteed performance for large flows.

The interaction between humans and computers will be greatly enhanced by the integration of multimedia technology with existing computer systems. We are seeing major developments in the areas of multimedia hardware devices and applications. However, the design of system software at the operating system and network software levels remains a challenge. This is particularly true when support for digital continuous media is required. High-bandwidth, low-delay guarantees are needed for communication, and operating system and network software must work synergistically to achieve the provision of such guarantees all the way up to the application layer.

References

1. G. H. Domann, ‘‘B-ISDN,’’ *Journal of Lightwave Technology* **6**(11), pp. 1720-1727 ().

2. D. P. Anderson, R. Govindan, and G. Homsy, ‘‘Abstractions for Continuous Media in a Network Window System,’’ Report No. UCB/CSD 90/596, Berkeley, CA (September 1990).

3. D. Ferrari, ‘‘Client Requirements for Real-Time Communication Services,’’ *IEEE Communications Magazine,* **28**(11), pp. 65-72 (November 1990).

4. G. Karlsson and M. Vetterli, ‘‘Packet Video and Its Integration into the Network Architecture,’’ *IEEE Journal on Selected Areas in Communications,* **7**(5), pp. 739-751 (June 1989).

5. C. Nicolaou, ‘‘An Architecture for Real-Time Multimedia Communication Systems,’’ *IEEE Journal on Selected Areas in Communications,* **8**(3), pp. 391-400 (April 1990).

6. W. H. Leung, T. J. Baumgartner, Y. H. Hwang, M. J. Morgan, and S.-C. Tu, ‘‘A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks,’’ *IEEE Journal on Selected Areas in Communications,* **8**(3), pp. 380-390 (April 1990).

7. B. Leiner, Ed., ‘‘Critical Issues in High-Speed Networking,’’ RFC 1077, Network Information Center, SRI International, Menlo Park, CA (November 1988).

8. A. J. Frank, L. D. Wittie, and A. J. Bernstein, ‘‘Multicast Communication on Network Computers,’’ *IEEE Software* **2**(3), pp. 49-61 (1985).

9. S. E. Deering and D. R. Cheriton, ‘‘Multicast Routing in Internetworks and Extended LANs,’’ *ACM Transactions on Computer Systems* **8**(2), pp. 85-110 (May 1990).

10. D. P. Anderson, S.-Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, *Support for Continuous Media in the DASH System,* University of California, Berkeley, Berkeley, CA (October 16, 1989).

11. T. D. C. Little and A. Ghafoor, ‘‘Synchronization and Storage Models for Multimedia Objects,’’ *IEEE Journal on Selected Areas in Communications,* **8**(3), pp. 413-427 (April 1990).

12. D. Ferrari and D. C. Verma, ‘‘A Scheme for Real-Time Channel Establishment in Wide-Area Networks,’’ *IEEE Journal on Selected Areas in Communications,* **8**(3), pp. 368-379 (April 1990).

13. C. Topolcic, Ed., ‘‘Experimental Internet Stream Protocol, Version 2 (ST-II),’’ RFC 1190, Network Information Center, SRI International, Menlo Park, CA (October 1990).

14. D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, ‘‘An Analysis of TCP Processing Overhead,’’ *IEEE Communications Magazine* **27**(6), pp. 23-29 (June 1989).

15. W. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meister, H. Rudin, and R. Williamson, ‘‘A Survey of Light-Weight Transport Protocols for High-Speed Networks,’’ *IEEE Transactions on Communications* **38**(11), pp. 2025-2039 (November 1990).

16. B. Arons, C. Binding, K. Lantz, and C. Schmandt, ‘‘The VOX Audio Server,’’ *Proc. Multimedia ’89,* (April 20-23, 1989).

17. D. M. Ritchie, ‘‘A Stream Input-Output System,’’ *AT&T Bell Laboratories Technical Journal,* **63**(8), pp. 1897-1910 (October 1984).

18. M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, ‘‘Mach: A New Kernel Foundation for UNIX Development,’’ *Proceedings of the Summer USENIX,* (July, 1986).

19. D. D. Clark and D. L. Tennenhouse, ‘‘Architectural Considerations for a New Generation of Protocols,’’ *ACM SIGCOMM ’90,* pp. 200-208 (September 24-27, 1990).