Video-server designs for supporting very large numbers of concurrent users

by M. Kumar

We present a design for a video server in which video content is stored in special video-delivery subsystems attached directly to network components such as switches or high-speed-network ports rather than the magnetic disk storage attached to conventional computer systems. Video is preformatted and stored in the form of network packets. This design approach overcomes the CPU- and I/O-bandwidth limitations of conventional computers in executing the file-system and network-protocol code for many concurrent video streams, resulting in higher performance at a lower cost. Two designs for the approach are discussed. The first extends the packet buffer of a shared-buffer switch with additional memory for storing the video packets. The second design uses a stream controller as the interface between an array of disks and a traditional switch or network port. We have built a prototype based on the second design. To avoid interference on the disks, data is interleaved across all disks connected to a stream controller in units of fixed playback time. This also reduces the jitter in the response time of the disks and, therefore, the size of the buffers needed to maintain interruption-free delivery. The cost benefits of both approaches are discussed.

Introduction

It is widely believed that advances in computer and communication technologies will make possible a wide variety of new residential and commercial interactive multimedia services. The contemplated residential services include video on demand for movies, news, sports, TV programs, etc., home shopping, interactive games, surrogate travel, and a wide variety of educational and information services. The commercial services include video mail, conference record-keeping, multimedia manuals, training, and industry-specific uses such as videos of homes for sale, used in the real estate industry, and videos of vacation resorts for the travel industry.

0018-8646/98/$5.00 © 1998 IBM

Video servers are the specialized systems used by the providers of the services mentioned above to store the vast amount of video needed and to deliver it to individual users when requested by them.

Fiber optic technology, one example of these advances, has permitted upgrades to cable TV trunk and feeder systems, allowing each active subscriber to have a dedicated channel to a server for receiving compressed digital video. Personal computers and set-top boxes have evolved to support networked multimedia applications, taking advantage of low-cost video-decompression chips and network-interface chips for the cable network.

The current choice of video servers for interactive multimedia services, however, continues to be standard mainframes or workstation-based parallel/clustered computing systems. Their system architecture, hardware organization, operating systems, and I/O subsystems are not matched to the requirements associated with delivering multimedia content to a large number of networked users. Mainframe and workstation hardware is optimized for processing computation-intensive applications, with very limited emphasis on moving data efficiently between the network interfaces and storage devices, which is the primary requirement for a video server. For example, the bandwidth between the memory and cache in standard systems is an order of magnitude higher than the bandwidth between the memory and the storage or network devices. The floating-point units add to the cost of the systems without providing any benefit to the delivery of multimedia data, and the caches are too small to capture any locality in the accesses to multimedia data.

Similarly, most operating systems of mainframes or parallel-computer-based servers are optimized to maximize utilization of the CPU and to maximize throughput in a time-sharing environment. The response time for an operating system service can vary significantly from request to request. Therefore, large buffers in system memory are needed for multimedia data being retrieved from secondary storage. Finally, the disk-array subsystem of a standard system is itself usually optimized to retrieve data called for by a single I/O request with the minimum latency and highest bandwidth possible. The high bandwidth is achieved by reading large blocks of data from the disk array, which creates the need for large semiconductor storage buffers if many multimedia streams have to be provided simultaneously. This increases the total cost of the server.

The aforementioned limitations in using general-purpose computers as video servers force the price/performance of such servers to be much higher than that of a system designed optimally for delivery of video. (Of course, this is true only if the design and product-development costs of the special-purpose system can be apportioned over a sufficiently large number of units.) Several researchers have addressed the operating-system drawbacks by fine-tuning the operating-system services. Dan and Sitaram have developed innovative methods for optimizing the use of stream buffers in the system memory of a general-purpose computer being used as a video server [1, 2]. Haskin has proposed increasing the block size in the file system to improve performance [3]. Rangan and Vin have suggested novel techniques for placement of data on disks [4, 5]. Several researchers have developed improvements in scheduling disk read commands [6–8]. Tobagi et al. have proposed a real-time kernel-based system in which a periodic process schedules the retrieval of multimedia data from a disk array [9]. Serpanos and Bouloutas have compared the performance of video servers based on conventional computers configured as centralized and distributed systems [10]. A comprehensive tutorial on the design issues in digital multimedia servers is given in [11].

In this paper, we describe a radically different approach for delivering video over a network from a centralized server to a large number of clients. We observe that much of the processing done in a traditional video server to retrieve data from a file system and reformat it into network packets is wasteful because video files stored in the server are sent repeatedly at different times over the network, with a nearly identical sequence of packets being created each time, the only difference being that they are addressed to different recipients. A logical approach to reducing the processing requirement, hence the cost of the server, is to create this sequence of packets once and store it in memory coupled closely to the network, so that the packets can be retrieved much less expensively than with a conventional file system and delivered to the network without incurring the network-protocol overhead more than once. This reduces the processing required to send video from a few instructions per byte transmitted to a few instructions per packet transmitted, the exact number of instructions depending on the cleverness of the implementation in both cases.

In the next section of this paper, we present a video-server architecture that embodies the above philosophy of preformatting data into packets and storing them in memory closely coupled to the network. The storage for video packets is integrated directly into a network switch. This architecture is suitable for high-end video servers providing tens of thousands of streams, which is an order of magnitude higher than what can be accomplished with general-purpose-computer-based servers with similar hardware complexity. The design of large switches is optimized to move data at high speeds from the switch inputs to the outputs via intermediate buffers, though the switches cannot do any significant processing on the data passing through them. By extending the internal buffers of

the switch to enable them to store video and minimizing the processing required to create network packets by preformatting the stored video as network packets, one can transform a basic switch into a video server that is much more cost-effective than a general-purpose computer.

Though we have worked out the design and implementation details of this architecture, it has not been implemented, in part because the perceived demand for such large servers was revised sharply downward during the design period (a few months in 1992). Instead, we built a smaller server, primarily a software rendering of the design philosophy. In this design, we store video in the format of network packets in a disk array that is coupled via a stream controller directly to network interface cards in a workstation. This design is described in the third section of this paper. It is applicable to servers providing several hundred streams.

The stream controller is capable of completing the recipient address in the stored packets. The packets of a video file are linked together. Therefore, the stream controller can deliver packets directly to the network without the intervention of a general-purpose computer. Buddhikot et al. proposed a system for directly delivering multimedia data from an array of disks to an ATM network [12], but, as is the case with RAID II [13] (Redundant Array of Inexpensive Disks), a general-purpose computer is still responsible for initiating every data transfer from the disk. Therefore, the scalability of their approach is limited by the computer's processing capability.

In our system, data is interleaved[1] in units of fixed playback time, as proposed by Lougher and Shepherd [14] and Chang and Zakhor [15]. Therefore, we can use a software scheduler in the stream controller to coordinate the access to the disks by the video streams to completely avoid interference. This achieves not only higher throughput from the disks but also much tighter control on response time, thus avoiding jitter (the irregularity observed by the viewer or listener when video or audio packets are not received in time). Therefore, the size of the buffers required for a video stream can be reduced by nearly an order of magnitude, from 512 kilobytes (KB) or 1 megabyte (MB) to 64 or 128 KB. This reduces cost and is critical for implementing the stream controller so that the video-stream buffers are all on a single card that fits in a workstation adapter. If the video buffers must be in the system memory of a general-purpose processor instead, video data will move twice over the I/O bus, reducing the number of streams that can be delivered.

[1] Interleaving a file on d disks means writing successive portions of the file on successive disks, 1 to d, and repeating this until all of the file has been written. Then, each disk contains every dth portion of the file. Interleaving is also called "striping."

In the last section of this paper, we discuss the performance of the video-server prototype and comment on the importance of the proposed ideas in achieving the performance. Simulation results are presented to demonstrate the problem of disk interference, which limits the throughput of conventional servers, but is circumvented by our design. We also discuss the advantage of using semiconductor memory for large video servers that can serve tens of thousands of streams from a few dozen files. Briefly, when economics are determined for this situation, the total cost of storing the video must be divided by the number of streams that can be served. Surprisingly, DRAM is cheaper than disk storage because it has much higher bandwidth.

While we have focused on a very important requirement of video servers, namely their ability to move data from storage to network in large volume at low cost, a video server must provide other capabilities, such as accepting requests for starting new streams, authentication of authorized users, and billing systems. These are not covered in this paper but can be found in [16]. The proposed design is also applicable to large servers delivering information other than video and audio over networks. Nonvideo files in network file servers can be similarly prepacketized and off-loaded to specialized packet-delivery systems, especially if the files are large and are requested frequently.

Placing video data in network switches

Our video-server design is motivated by the following observation about the traditional implementation shown in **Figure 1(a)**. When thousands of streams access the same video material at slightly different times, the general-purpose processor fetches the data from the disk subsystem each time, executes the file-system code for that purpose, and then executes the network-protocol code to deliver the video. However, the result is essentially the same sequence of packets each time, differing minimally from the packets of the previous stream, perhaps only in the destination-address field of the network headers and trailers. The number of streams delivered is limited by the CPU's ability to perform these functions.

Since the video data for an application program is not processed or modified by the general-purpose processor running that application, it can be partitioned into video messages (for example, frames), which can be represented by pointers (addresses) in the memory of the processor to the actual video message stored in a switch of the network. (In general, all of the messages of a video are stored in one switch. Copies of messages can be stored in different switches, for the sake of reliability or increased throughput.) When the processor must send video to an end user, it sends requests to the network switch storing that video to deliver the constituent video messages,

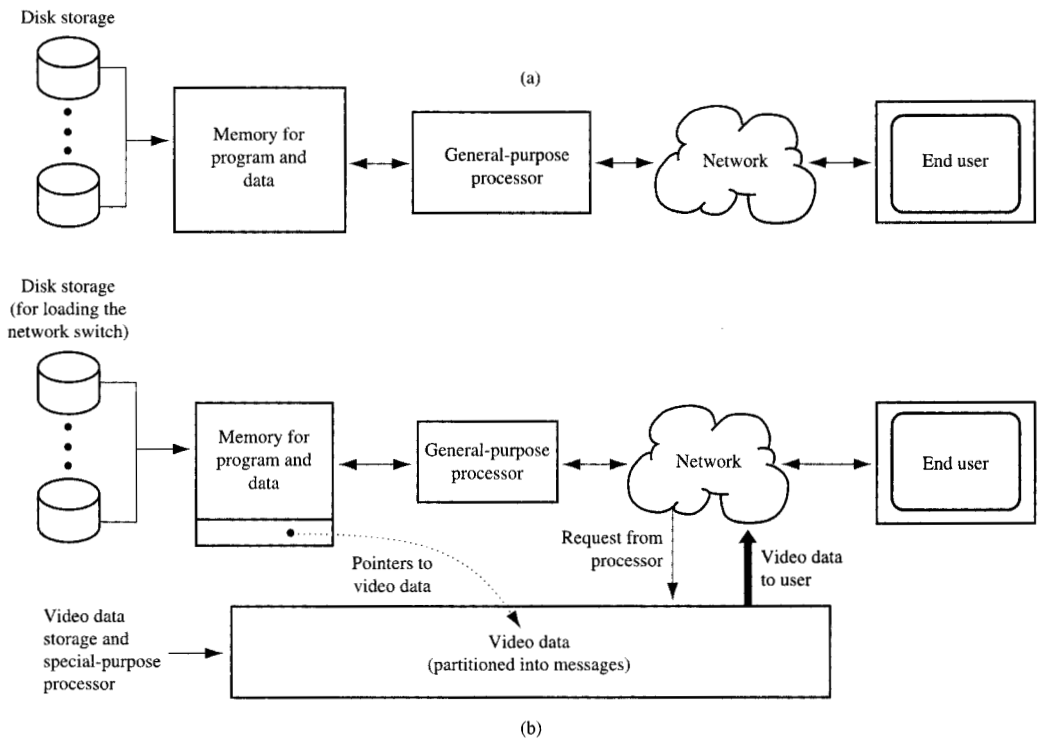

Figure 1

Multimedia design: (a) Traditional approach, using general-purpose processor; (b) approach with video data in network switches, and using special-purpose processor.

instead of reading video from its own secondary storage. The processor sends one request for each message into which the video is partitioned, or, as discussed below, the processor may send only one request for a group of video messages that are connected by links. A request consists of a pointer to the video messages, the address of the recipient, and possibly the time at which the video message should be read out. **Figure 1(b)** illustrates this concept. Thus, we eliminate the recurring overhead of generating nearly identical video messages for delivering the same video material to multiple viewers. Furthermore, instead of using the expensive computational resources of a general-purpose (host) computer, we use less expensive dedicated hardware (special-purpose processor) integrated into the switches of the network.

Before the video messages are stored in the switch, transport-layer headers and trailers are attached to the messages, the messages are segmented into network-packet payloads, and network-layer headers and trailers are attached to these payloads. The fields in the network- or transport-layer headers or trailers that can be precomputed when the packets are stored in the switch are so computed and stored in the network packets. The fields that cannot be precomputed (e.g., recipient) are completed subsequently, as described below. By storing these preprocessed packets in the network switch, we further reduce the processing required in the switch to deliver video data.

A video message is the smallest unit of video that can be sent to a client. The size of the video message has to be small enough to provide quick response time in interactive multimedia applications (e.g., to stop the play of a video or to switch to a new stream). However, short video messages require more frequent interactions between the general-purpose processor and the switches, demanding more powerful (therefore more expensive) processors. Furthermore, more hardware is required in the switches to handle the increased number of request messages. In order to minimize both message size and general-purpose processor and switch processing, successive video messages of a video file are connected by a link field. Thus, a request from the general-purpose processor identifying the first video message in the video stream and the number of subsequent messages to be

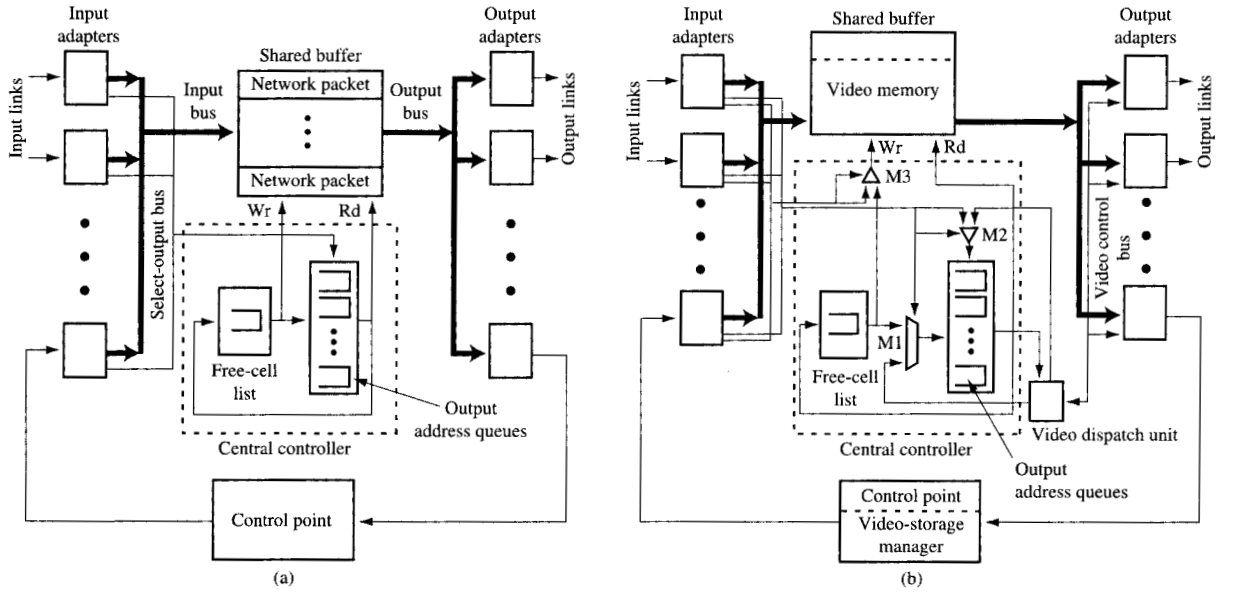

Figure 2

Shared-buffer switch: (a) Conventional design; (b) augmented in order to store video data.

delivered enables the switch to deliver all of the specified messages in the stream without further processor intervention. We expect the message size to be between 16 KB and 32 KB. The link field, in addition to carrying a pointer to the next message in the video storage, can optionally include a time value, expressed as an offset from the video start time. When a link field contains a time value, the video message to which the link points is requested at the specified time. Otherwise, that message is requested immediately.

The video messages can contain multiple links, in order to provide access to video data packets in sequences different from those requested for normal playback. For example, for video compressed using the MPEG** (Moving Picture Expert Group) algorithm [17], all of the I-frames can be linked together as a doubly linked list for providing fast-forward and reverse modes. Additional time values can be included for supporting slow motion. Finally, links that point to syntactic boundaries such as the start of a scene or the start of a spoken sentence can be generated and used for sophisticated forward- or backward-motion functions.

Shared-buffer switch modified to store video

In this section, we first briefly discuss the design of a conventional shared-buffer switch and its operation. Then we describe the modifications that give the switch the capability to store video and dispatch specified video packets to specified clients when instructed by the general-purpose processor. Such switches are used in high-speed packet-switched networks to move packets from one link to the next on their path to the destination.

Figure 2(a) shows the high-level architecture of a conventional shared-buffer switch. At its core is a large shared buffer, with an input bus for receiving packets into the memory from incoming switch links, and an output bus for sending packets from the memory to output links. The bandwidths of the input and output buses are equal to the aggregate bandwidth of all the input and output links, respectively. A list of free cells (fixed-size blocks) in the shared buffer is maintained in a FIFO queue in the central controller. Each output link also has an address queue associated with it in the central controller, for storing the addresses in the shared buffer of the network packets that must be delivered to it.

Network packets are transmitted through such a switch as follows. The network packets arriving on each input link are processed in the input adapter to determine to which switch output the packet must be routed, and the selected switch output is signaled on the *select-output* bus. In order to store the incoming packet into the shared buffer, the address of a free cell must be obtained from the list of free cells, placed on the write address bus of the shared buffer (labeled Wr in the figure), removed

from the list of free cells, and placed in the address queue of the output link to which the packet is being sent. Each output adapter dequeues the addresses of packets in the shared buffer from its address queue in the central controller, reads the packets over the output bus, and sends them on its output link. The address is stored back into the list of free cells.

The input bus is operated in a slotted manner (thus, no arbitration is required), with each of the N input adapters accessing the bus every Nth time slot. This is possible because the bus bandwidth is N times that of the fastest input link. The output bus is operated in a similar manner, and each input and output adapter interacts with the central controller only in the clock cycle in which it gains access to the input or output bus. A microprocessor in each input and output adapter is used to perform various link-monitoring and service functions.

The control point in Figure 2 is a general-purpose computer that performs network-management functions.

Figure 2(b) illustrates a conventional shared-buffer switch with the hardware modifications required to give it the capability to store video messages and to deliver a specified group of video messages to a specified client when instructed by the general-purpose processor to do so. In order that network packets of a video message can be stored in the shared-buffer switch of Figure 2(b), the shared buffer is augmented with video memory. (The shared buffer is used for regular network traffic, with packets arriving and departing. The video memory is used to store video packets, which remain until explicitly deleted.) The shared buffer and the video memory share the input and output buses, the write address bus (labeled Wr), and the write-enable control (not shown in the figure). The link field and other auxiliary fields associated with the video messages are stored separately in a tag memory, which resides in the video dispatch unit.

The output adapters perform the function of managing the video streams originating from video memory. Unlike the fields in the packets coming from the shared memory of a conventional shared-buffer switch, the destination address fields and some other fields of the packets coming from the video memory do not contain the proper values; however, the microprocessors in the output adapters are programmed to complete the headers and trailers of the packets received from video memory before forwarding them. The command sent (in a packet) from the general-purpose computer [Figure 1(b)] for dispatching a video message is intercepted by the output adapter through which the messages will be transmitted. The microprocessor in the output adapter maintains a stream-control table with an entry for each video stream it controls. The video-service application running in the general-purpose computer manipulates the stream-control table through control messages sent to the output adapters, which use these tables to send requests to the video dispatch unit to deliver individual video messages. The video dispatch unit, in turn, requests the central controller of the switch to deliver all of the network packets in the message, by issuing requests for one packet at a time to the central controller. The output adapters interact with the video dispatch unit via the video control bus in Figure 2(b). As with the central controller, each input and output adapter interacts with the video dispatch unit only in the cycle in which it accesses the input bus or the output bus; therefore, no arbitration is required for the video control bus.

The video dispatch unit also monitors the addresses of all cells read from the video memory, and the video streams to which they belong. If the packet read from the video memory is not the last packet of a video message, the video dispatch unit generates a request for the next packet in the video message by reading the link field of the current packet and sending it to the central controller. If the packet read from the video memory is the last packet of a message, the link field of that message (which contains the address of the next video message in the stream and, optionally, the time at which the next message should be delivered to the client) is read from the tag memory in the video dispatch unit and sent to the output adapter receiving the video message. The link field is used by the microprocessor in the output adapter to send a request for a new video message to the video dispatch unit or to notify the general-purpose processor (by means of a packet sent from the output adapter to the processor) of completion of the video message delivery.

So that it can interact with the video dispatch unit and with the input adapters (to load video content), the central controller is modified as follows. A multiplexer unit, M1, is provided between the free-cell list and the output-address queues, in order to allow the video dispatch unit to add *stream_id*s to the packet addresses stored in the address queues. When a packet address is dequeued from the output-address queue, the video dispatch unit uses the *stream_id* part to determine which video stream it belongs to and, consequently, the video stream for which it must perform further activity, such as completing the information in the header and/or trailer fields of the packets. Another multiplexer, M2, allows the address queue to be selected by the video dispatch unit when it is writing the address of a packet in video memory into an address queue. Addresses dequeued from address queues are recycled into the free-cell list only for cells associated with the shared buffer, not with video memory.

- *Loading video memory from the general-purpose processor*

To load information into the video memory, the microprocessor in an input adapter receives messages, from the general-purpose processor, comprising the

network packets to be loaded and their starting addresses in the video memory. In response, the input adapters place the packets on the input bus and place the addresses in video memory where the packets should be stored on a separate bus connected to the third new multiplexer, M3, in Figure 2(b). Multiplexer M3 selects the write address for the shared buffer and the video memory. The general-purpose computer used as the control point in the shared-buffer switch also functions as the video-memory manager, to allocate and reclaim video storage. The general-purpose processor or application server interacts with the video-memory manager (by means of command packets) to request a block of free video memory or to return a block of video memory. Once the video-memory manager allocates a block of video memory to the general-purpose processor and provides the processor with the address range for that block, the processor can request an input adapter to write directly into the allocated video memory, as explained in the preceding paragraph, without further involvement from the video-memory manager.

Stream controller to attach disks to a switch

In the previous section, we presented a high-level design for storing video content in the switches of a network. An alternative to this approach is to store video in a separate subsystem that can be connected to the switches, or alternatively, connected to a general-purpose computer to augment its video-delivery capability.

In this section, we first review the design of the stream controller, a hardware unit designed to attach an array of disks directly to an input port of a switch. Several such stream controllers, each controlling a separate array of disks, can be connected to different input ports of a switch in order to increase the capacity of a server, both in terms of amount of storage and the number of streams delivered concurrently. Then we describe the format for storing video data on disks as network packets collected into disk-access units called groups of blocks (GOBs), the GOBs of a video stream being connected by link fields. The GOBs have fixed playback time, which allows us to use the disk bandwidth more efficiently.

- *Stream-controller design*

The function of the stream controller is to retrieve the network packets of video clips stored on disks and send them to the switch inputs. The design is optimized for handling a large number of low-bandwidth I/O transfers concurrently, which is required for video servers. In contrast, traditional RAID systems are optimized to support infrequent high-bandwidth transfers.

The design of the stream controller is shown in **Figure 3**. At its core is the stream-buffer memory, constructed from semiconductor storage and organized as a two-port memory. One port, labeled in the figure as the disk port,

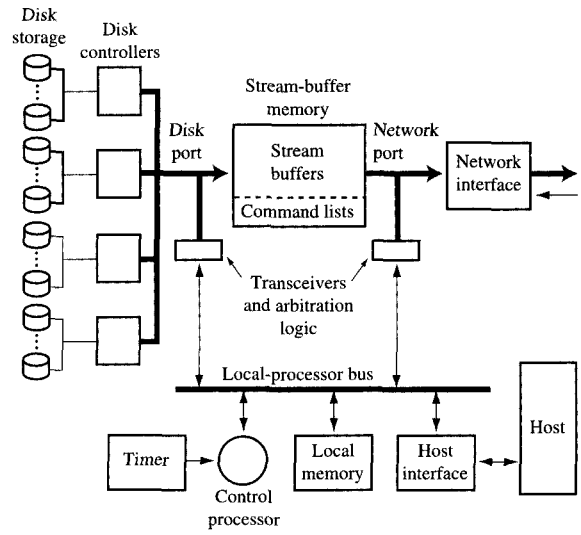

Figure 3

Block diagram of the stream controller.

is used to receive data from the disks. The disk port of the stream-buffer memory has interfaces to several disk controllers, each connected to several disks. The other port, labeled as the network port, is used predominantly to read network packets for delivery to the network. The network port of the stream-buffer memory is connected to the input of a switch through the network interface logic. The stream-buffer memory is partitioned into multiple stream buffers, two of which are allotted to each active video stream when the video stream is set up (created). A real-time control processor, typically an off-the-shelf microprocessor, is connected to its local memory through the local processor bus. An Ethernet or RS/232 interface allows a remote general-purpose processor to perform setup and service functions, such as downloading programs for the control processor to its local memory and monitoring error logs. The transceivers and arbitration logic, shown in the figure, allow the control processor to access the stream buffers, the control registers in the disk controllers, and the control registers and storage in the network interface. They also allow the network interface logic and the disk controllers to interrupt the control processor and access its local memory.

The timer provides periodic interrupts to the control processor in the stream controller. In response to these interrupts, the control processor sets up the disk controllers to transfer prepacketized video data from their disks to the stream buffers. It writes a list of read commands for each disk controller in an area of the

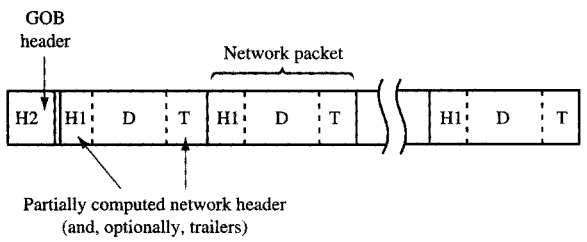

Figure 4

Internal structure and format of a GOB (Group Of Blocks), the unit of interleaving video data on disks.

stream-buffer memory reserved for that purpose and passes to the disk controller a pointer to the command list. The disk controllers indicate the completion of the read commands by interrupting the control processor.

The missing fields in the network header and/or trailer are computed by the control processor and updated in the stream buffer, and the completed packet is forwarded to the network-interface logic. The network-interface logic also receives a command list from the control processor, each command in which specifies the address in the stream buffer of a network packet to be transmitted to the switch, the size of the packet, and optionally a header that should be appended to the packet. Similar to the disk-controller command lists, the network-interface command lists are also written by the control processor into the stream buffer, and the starting address and size of the command list are written into the control memory of the network interface logic.

We propose to connect the stream controller to the switch inputs in order to avoid design changes in the switch. In the case of shared-buffer switches, one could alternatively couple the network interface of the stream controller directly to the shared buffer in the switch. Though the latter approach saves some hardware (switch and input adapters in the switches and network interface logic in the stream controller), it requires redesign of the switch to accommodate this connection.

• *Optimized format for storing multimedia data on disks*
The compressed multimedia data is stored in the disk array in the form of network packets in order to reduce the amount of protocol processing carried out in the stream controller. If video data is stored in the form of IP/UDP (Internet Protocol/User Datagram Protocol) packets, the IP and UDP headers are precomputed (except for the destination address and port numbers) at the time the video data is loaded in the disk array. When the packets are scheduled to be sent out to the network interface, the destination addresses and port numbers are filled in and the checksums are modified as required. Note that much less CPU time is required to do this than to compute the entire header and checksum "from scratch." This technique is equally applicable to ATM networks. For example, the entire ATM AAL5 (Adaptation Layer 5 [18]) convergence sublayer can be precomputed at the time video is stored on the disk.

The size of the network packets stored on the disks is typically in the range of 512 bytes to 4 KB. Reading one packet at a time from the disk will result in inefficient use of the disk bandwidth. Therefore, several network packets are combined into a single disk-access unit, which we call a group of blocks (GOB) of size 64 KB to 256 KB. A GOB is also the basic unit of disk-storage allocation. The format of a GOB is shown in **Figure 4** and is discussed below.

The compressed video data (e.g., compressed by the MPEG algorithm) stored in a GOB has a predetermined playback time, fixed for all the GOBs of a stream controller. For multimedia data compressed at a constant bit rate (i.e., a constant number of bits per second of video), all GOBs in a stream controller have the same size. However, if multimedia data is compressed at a variable bit rate, as was the case with our prototype, the GOBs have varying numbers of disk blocks. While the disk controller accesses data from the disk one GOB at a time, a GOB is too large to be sent to a client in a single burst; thus, the GOB is further divided into n clusters of packets of equal playback duration and, therefore, of possibly different sizes.

Each video file is interleaved across all disks of a stream controller, with each GOB having fixed playback time. Hence, the multimedia video streams "hop" from one disk to the next in one disk cycle (the GOB playback period) when fetching GOBs, with all streams in synchronism. That is, all streams fetching a GOB from a disk i in one disk cycle will fetch their next GOBs from disk $i + 1$ (modulo the number of disks) during the next cycle. If a new video stream can be added to a group of streams simultaneously accessing a disk without overloading the disk, the expanded group will move from one disk to the next together, accessing consecutive disks in consecutive disk cycles without overloading them. (We have assumed that all disks have identical throughput. We have further assumed that there is a reasonable upper bound on the size of a GOB, so that we can guarantee the maximum number of GOBs that can be read from a disk.)

Interference on the disks is eliminated by using a schedule table such as **Table 1**. It has one column for each disk controlled by the stream controller and shows the relative position of streams with respect to one another. Stream S_1 is arbitrarily assigned to the first disk in the

first row. Thus, the schedule table shows the position of each stream (the disk it is accessing) when stream 1 is accessing disk 1. In Table 1, for example, streams S_1, S_3, and S_4 access disk 1, while streams S_{11} and S_{12} access disk 2, and so on. In the next disk cycle, S_1, S_3, and S_4 access disk 2, while S_{11} and S_{12} access disk 3, and so on. To start a new stream S_n, the scheduler in the general-purpose computer simply finds an empty cell in the schedule table and enters the stream identifier in it. Suppose the cell is in column i. The new stream will always stay $i - 1$ streams ahead of stream S_1, accessing disks with other streams in column i. The number of rows in the schedule table equals the maximum number of streams that can read a GOB from one disk in a disk cycle (three in Table 1). The above scheduling policy can be extended to optimally handle video files that are played back at widely varying bit rates. These extensions and the modifications required to support "trick" playback modes, such as fast-forward, rewind, and slow motion, are discussed in [19]. To handle trick playback modes and keep response times low, five or ten percent of the cells in the schedule table are not allocated by the scheduler. The freed bandwidth is also used for loading new content into the disk arrays managed by the stream controller. We could also keep GOB sizes in the GOB header, which would allow the stream controller to determine whether the time required to read all of the GOBs scheduled to be read in a disk cycle would be small enough to permit disk access for a stream playing in trick mode or for downloading content.

The remote general-purpose processor allocates the storage for multimedia data in chunks of several hundred megabytes and partitions it into GOBs having the fixed playback time used by the stream controller, by analyzing the data as it is being stored there. Each GOB is referenced by a GOB pointer (in both the general-purpose computer and the link fields stored in other GOBs) comprising the disk number, the starting block on the disk, and the number of blocks in the GOB.

While each packet in a GOB has its own network header (H1 in Figure 4) and possibly a trailer, the entire GOB has a GOB header (H2), which includes the GOB pointer to the next GOB of the same multimedia stream, thus enabling the stream controller to autonomously access consecutive GOBs of a multimedia stream. For the sake of checking integrity of the stored video data, we also included in the GOB header a pointer to the preceding GOB of the same video stream. If fault tolerance is implemented by storing a parity GOB for a group of data GOBs, the GOB header includes pointers to all GOBs in the parity group of the next GOB. Finally, the GOB header also contains pointers to all of the clusters of packets in the GOB.

Table 1 Example schedule table for a disk array with M disks. The entries in column j are the streams accessing disk j in the first disk cycle. In this example, three GOBs can be read from each disk in a disk cycle.

Disk number				
1	*2*	*3*	...	*M*
S_1		S_2		S_7
S_3	S_{12}	S_5		S_8
S_4	S_{11}			S_9

For situations in which a large number of different network protocols must be supported from the same video content, it would be more efficient to store the network-packet payloads and packet headers and trailers as separate arrays in the GOB. For a single array of packet payloads, one can have a different array of network headers and/or trailers to support each protocol. The control processor would then place separate entries for the header, trailer, and payload components of the network packet in the command list sent to the network interface. The network interface, usually capable of performing linked DMA, would assemble the packet from the two or three entries. If different video clients, even when using the same protocol, must be sent packets of different size because of peculiarities of their connection to the video server, network packet payloads would again be stored separately from the headers, as above. The payload size would be the maximum expected. The stream controller would fragment the packet by replicating the header, just as routers do today.

- *Stream-controller software*

The key software running on the control processor in the stream controller consists of two real-time processes, the disk-read process and the network-transmit process, which transfer data from the disks to stream buffers and from there to the network. The disk read is a periodic process, each period being equal to the GOB playback time (disk cycle) for the stream controller. In each GOB playback period, the disk-read process issues a read command for each active multimedia stream, in order to retrieve a GOB of that stream from a disk. Each GOB playback period is further divided into n network cycles, and the network-transmit process is executed once every network cycle for each active multimedia stream, in order to transmit a cluster of packets from the stream buffer to the network. In our prototype, timer-generated interrupts define the disk and network cycles and initiate the disk-read and network-transmit processes.

A stream-control table regulates the disk-read and network-transmit processes. It maintains the state for each active stream, which includes the following entries or

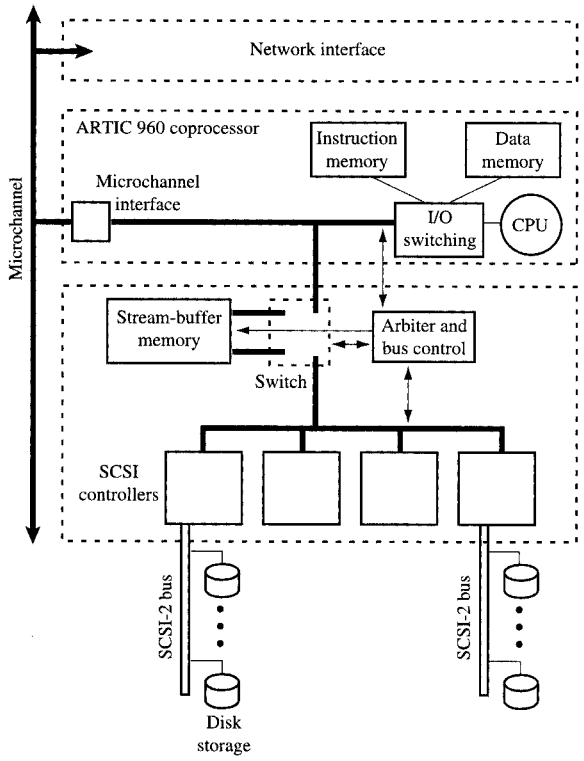

Figure 5

First prototype of the stream controller.

fields: pointers to two stream buffers in the stream-buffer memory, one to the read buffer that is receiving data for that stream from the disks and the other to a previously filled buffer that holds data being transmitted to the network; a pointer to the GOB being transferred from the disk to the stream buffer; a playback-mode field indicating whether the stream is in normal playback mode, paused, or in fast-forward or rewind mode; the client-information field, which points to the block of information needed to complete the network-protocol processing on the network packets retrieved from the disk (for example, for IP networks, it would contain the destination IP address and UDP port number for the video stream); and a field that specifies the stopping condition for the stream, either as the pointer to the last GOB to be played back or the time remaining until the completion of playback.

The stream-controller software also includes a non-real-time process that presents a VCR-like interface to the remote general-purpose processor and allocates the stream-controller resources to a multimedia video stream being created or releases the resources of a stream being terminated. Service processes for loading data from the remote processor to the disks managed by the stream controller, allowing the processor to check various data structures in the stream controller and to verify the contents of the disks, are also part of the stream-controller software.

- *Stream-controller prototype*

The stream controller in our first prototype is a set of three Micro Channel* adapter cards, which plug into an RS/6000* workstation, as shown in **Figure 5**. One card has the network-interface logic, the second has the control processor and its local memory, and the third has the stream-buffer memory and four SCSI controllers, which provide four fast, wide SCSI buses, each having a peak transfer rate of 20 MB/s. The control processor is an off-the-shelf ARTIC 960 card running a real-time operating-system kernel. The network interface is also an off-the-shelf card. We designed and built the card holding the stream buffers and SCSI controllers because one with the required throughput and functionality was not available in 1993. If the stream controller were reimplemented today, this piece of hardware probably could also be an off-the-shelf component.

The prototype stream controller is capable of serving 250 MPEG-1 digital compressed-video streams, which requires video to be delivered at a sustained total bandwidth of 48 MB/s. Therefore, each SCSI bus has to sustain a transfer rate of 12 MB/s, which is achieved by connecting eight disks to each SCSI bus. The disks are 3.5-in. SCSI disks, each capable of storing 2 gigabytes (GB) of data. In our prototype, the GOB playback time is 200 milliseconds, which corresponds to approximately 64 KB, roughly the size of a track on modern magnetic disks. At that GOB size, these disks easily sustain the required bandwidth of 1.5 MB/s, especially since the queued read requests are reordered automatically in the disk controller to minimize seek latency. Details about the implementation of this prototype can be found in [19].

To support MPEG-2 streams at 4 Mb/s (megabits per second), we would choose 100-millisecond disk cycles, and the GOBs would be approximately 50 KB. Three GOBs can be read from a disk in each disk cycle, allowing 96 MPEG-2 streams to be supported from the 32 disks attached to the stream controller.

The above prototype has been implemented in 1993 technology. If the design were to be reimplemented today, much more powerful stream controllers could be built at a lower cost. Not only are more powerful microprocessors and denser memory modules available at a lower cost, but the speed of I/O buses has increased, and improvements in packaging technology allow us to place more components in available card areas.

• *System-design issues*

The maximum performance obtainable from any implementation of the proposed design is limited by the bandwidth of the connection between the stream controller and the network. To achieve this performance, the video buses between the stream-buffer memory and the disk controllers must be able to provide the same bandwidth. The number of disk controllers is chosen to meet or exceed this bandwidth. The aggregate bandwidth into and out of the stream-buffer memory must exceed the bandwidth of the video buses, the excess needed to allow the control processor, disk controller, and network-interface logic to exchange command lists.

For a fixed number of streams, a larger stream-buffer memory allows one to choose larger GOB sizes, which results in better utilization of the bandwidth of individual disks and longer disk cycles. Longer disk cycles reduce the performance required of the control processor. Since the video buses into and out of the stream-buffer memory operate near peak utilization, it is important to design the arbitration logic to ensure that the control processor has the necessary access to the stream-buffer memory in order to complete its chores in a timely manner. Other than these normal design issues, we did not encounter any major design problems.

Comparison of the performance of the proposed video storage method with the conventional approach

In this section, we first discuss the advantage of the proposed video storage format and the algorithm to schedule disk-read requests over the conventional method. While the proposed method is strictly deterministic, and its performance can be projected accurately, the conventional approach is stochastic, and simulations were used to predict its performance. Next we discuss the reductions in protocol-processing overheads achieved by storing data in the format of network packets. At the end of this section, we discuss the situations for which using semiconductor storage to store video results in a less costly solution than the use of disks.

• *Advantage of interleaving video in fixed playback units*

Simulation of conventional video server

We simulated the retrieval of video from an array of disks attached to a general-purpose computer in a conventional manner. We modeled 64 4GB disks, each completing a pending read request for a 256KB block in 100 milliseconds. The 256KB blocks retrieved from the disk were placed into a stream buffer, from which they were sent to the clients in five equal bursts spaced 100 milliseconds apart. Thus, the maximum possible throughput per disk is 2.56 MB/s, and video content is modeled as a constant-bit-rate stream compressed at 512 KB/s. The load on the disks was varied by varying the number of active streams. When fully loaded, each disk in the array would have an average of five active streams. While the model is very simple, it highlights the problem inherent in stochastic loading of disks.

Video content was modeled as 30-minute clips; hence, 283 clips could be stored on the disk array. The 256KB blocks of the video clips were mapped at random on the disk blocks. Thus, disk interference occurs when two or more video streams accessing different video clips, or even different portions of the same video clip, simultaneously attempt to access the same disk. [One could interleave these blocks on the disk in a round-robin manner. However, if the video files use variable-bit-rate compression, or if the individual files are compressed at a fixed rate but different files have different compression rates (a mix of MPEG-1, MPEG-2, and audio files compressed with different compression algorithms), the disk interference behavior is quite complex. Accurate modeling with round-robin placement would be difficult. Even if the modeling were accurate, the results would be valid only for the mix of files used. Using random placement is a good way of eliminating this complexity without affecting the results significantly.] Each stream selected one of the 283 video clips according to one of two models of popularity and then proceeded to access the blocks of that clip sequentially. The first model of popularity used equal probability for choosing any of the 283 streams. The second popularity model used the Ziff distribution, in which the probability of accessing clip i is

$$i^{-\alpha} \Big/ \sum_{i=1}^{283} i^{-\alpha}.$$

The exponent α was chosen to be 1.5 in order that the probability of accessing one of the ten most popular clips exceed 80% (actually, 81.11%). The stream-buffer sizes were varied from 512 KB to 4 MB. In order to start a stream, initial read requests were issued simultaneously, and the delivery of the stream to the network was delayed until a sufficient number of initial read requests to fill the stream buffer completely had been completed. From then on, a new read request for 256 KB of data was issued as soon as 256 KB of the stream buffer became free.

One of the quantities measured during simulation was the disruption count—the number of instances during the delivery of the video clip in which data could not be sent over the network because the stream buffer was empty. **Table 2** summarizes the disruption counts for 512KB and 1MB stream buffers for a uniform probability of selecting each video, and for 2MB and 4MB stream buffers for the

Table 2 Average number of disruptions per stream during a 30-minute video segment.

Load (%)	Popularity model			
	Uniform		Ziff	
	stream-buffer size		stream-buffer size	
	512 KB	1 MB	2 MB	4 MB
10	0	0	0	0
20	0	0	5	0
30	0.07	0	36	20
40	1.5	0	91	66
50	8	0	255	231
60	38	0.2	572	580
70	155	5.5	1113	1125

probability of selecting video being Ziff-distributed. One can readily see in Table 2 that if the popularity of the clips follows a Ziff distribution, five or more disruptions in a 30-minute period should be expected as soon as each disk has to support an average of more than one active stream (20% load) even though the stream buffers are 2 MB to 4 MB. When the popularity model follows a uniform distribution, five or more disruptions can be expected every half hour when an average of two active streams (40% load) are accessing each disk and the stream-buffer size is 512 KB, or when three active streams (60% load) are accessing each disk and the stream-buffer size is 1 MB.

On the other hand, the stream controller proposed in this paper would be able to support five streams per disk if the disk cycle were chosen to be 500 milliseconds and each stream had two 256KB buffers. Interleaving video in units of GOBs that have a fixed playback duration also gives us the flexibility to reduce the buffer size. In the above example, if we chose the disk cycle to be 100 milliseconds, the two buffers for each stream could be less than 64 KB each, because the disk response time can be guaranteed to be less than one GOB playback period. However, since we are now accessing disks for requests of smaller size, we can fit in only three requests in 100 milliseconds. This is still significantly better than the conventional approach, because three streams per disk are supported irrespective of the uneven access pattern, and with one tenth of the stream-buffer size. (Of course, there are no disruptions.)

• *Advantage of storing video in the format of network packets*

With prepacketization of the video stream, the i960 CA processor on the ARTIC 960 card (approximately 7-MIPS performance) was able to handle 80 streams with 30% processor utilization (even though the code was not fully optimized). The code required for header modification and preparation of the network DMA transmit list for each IP/UDP packet is well under 300 instructions. Consequently, we are confident that 250 MPEG-1 streams are achievable with the i960 processor. (We were not able to test the 250 streams because we did not have the needed stream-buffer memory.)

On the other hand, network-protocol-processing overheads and file-system overheads in conventional workstations are much higher. Kay and Pasquale [20, 21] have presented detailed measurements of various overheads for the UDP/IP protocol stack on a DECstation 5000/200 running the Ultrix 4.2a operating system. From the graphs presented in [20], we have determined that the overhead for checksum calculation, data movement, and protocol-specific operations limit the throughput to 3MB/s or, equivalently, 16 streams, as explained in [19]. In [22] Haskin and Schmuck describe a file system optimized for video delivery that can support 60 and 75 streams on an RS/6000 Model 970 and 980, respectively.

• *Economics and technical feasibility of using DRAM storage*

For serving a large number of streams, we propose the use of hundreds of gigabytes of semiconductor storage. An obvious concern is the technical feasibility and cost of building such a system. The media cost of disks (cost of the individual disks, excluding the electronics needed to build a system, power supplies, racks, markup, etc.), at about $0.15 per megabyte, is one fortieth of the media costs of DRAM, which is roughly $6 per megabyte. However, the maximum bandwidth of a 4GB disk is only 8 MB/s, while the bandwidth available from 4GB DRAM (organized in a configuration to provide maximum bandwidth) is 80 GB/s.

A server designed to provide 20 hours of popular programs and support 20000 simultaneous users would require 36 GB of storage (MPEG-2 video at 4 Mb/s) and 10 GB/s of bandwidth. While 36 GB of DRAM, costing $216000, would exceed the bandwidth requirement, 1250 disks, costing $750000, would be needed to provide the required bandwidth. Furthermore, in today's technology, 36 GB of DRAM can be packaged in a workstation-sized system, whereas 1250 disks would require at least 20 racks of equipment. Additionally, a large number of power supplies and disk controllers would be required to manage the disks, making the disk-based system even more expensive than the DRAM-based system. The operational costs and system management problems for the disk-based system would be larger.

Thus, even though disks are cheaper than DRAMs per bit of storage, the bandwidth constraints make the DRAM solution eventually cheaper for servers designed for large numbers of streams. Furthermore, if the content being

placed on the server can be partitioned into a limited-content, high-demand part and a large-content, low-demand part (a situation predicted in many applications), the cost of the overall system can be reduced significantly by placing the high-demand part in DRAM and the low-demand part on disks connected to the switch through stream controllers.

In the published literature, we have not seen any claims of substantial improvements in the execution of the Internet protocol stack, despite its importance in computer communication. Since it has existed for two decades, one might infer that opportunities for order-of-magnitude improvements are now unlikely. Of the many video-server trials mentioned in the trade press, only the work of Haskin et al. [3, 22] has been published in detail in the computer science technical literature. Because of the three-year hiatus between the two references and the indication of continued work by the authors on this subject during this time, we infer that [22] represents a near-optimum implementation, which would be hard to improve significantly. Hence, in our opinion it will be difficult to match the performance achieved by the proposed design approach. One should keep in mind that as hardware becomes faster because of improvements in semiconductor technology, both the conventional approach and the design proposed in this paper will benefit equally.

In the above analysis, we made the comparison for 20 hours of video. This design point was considered suitable for information and entertainment applications in which a user connects to the information or entertainment provider daily or weekly, and uses the service for thirty minutes to two hours a day. Presenting the user with a 20-hour selection for an average of one hour of viewing was considered adequate.

Summary and discussion

In this paper, we proposed a design for video servers in which video content is off-loaded from the magnetic storage attached to a conventional computer system and stored in special video-delivery subsystems attached directly to network switches. One video-delivery subsystem was a modified shared-buffer switch. Most other switch designs can be similarly modified by providing data paths from the video storage to all switch outputs and modifying the arbitration logic associated with each switch output port to accept the packets arriving from the video memory with higher priority than other traffic.

We also described an unconventional way of storing video content on a disk array. Though we proposed attaching the disk array directly to the network switches, as suggested in [19], the disks can be attached to a conventional computer as well. By simulating a simple model of a disk array, we showed the advantage over conventional disk-access methods of interleaving video data in units of fixed playback time and accessing it according to a fixed schedule. We also illustrated the reduction in protocol-processing overhead achieved by storing video content in the format of network packets with the precomputable parts of packet headers and trailers computed in advance and stored in the packets. We presented the economic rationale for using semiconductor storage for servers that have to support large numbers of streams with limited content.

An alternative to building a single large video server is to use many small servers, geographically distributed among the users. This has the advantage of reducing the network cost, but the content must be replicated. Furthermore, the operational cost of multiple-disk-based systems in residential neighborhoods would be higher than the operational cost of a centralized system. In our opinion, when video-on-demand systems become common, the centralized-server approach will be more economical.

In conclusion, we have demonstrated that multimedia servers architected and designed to optimize the delivery of video and other multimedia data will have a substantially better price/performance than general-purpose computers used for the purpose. However, these specialized servers will be economically viable only when the demand for them justifies the initial design and development costs.

Acknowledgments

Many people contributed to the design and implementation of the stream-controller prototype. Sneha Kasera wrote the real-time scheduler for the ARTIC 960 card. Mary McHugh provided the video formatting and placement software. Dan Fasano designed the data flow of the stream controller. Jack Kouloheris provided several hardware and software components and handled the integration and testing of these components. In the early phase of the project, discussions with Andy Lean, Jayanta Dey, and Chia-Shiang Shih helped shape the design of the stream controller.

References

1. A. Dan and D. Sitaram, "Buffer Management Policy for an On-Demand Video Server," *Research Report RC-19347*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1994.
2. A. Dan, D. Sitaram, and Perwez Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Research Report RC-19381*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1993.
3. R. Haskin, "The Shark Continuous Media File Server,"

Proceedings of IEEE COMPCON '93, San Francisco, 1993, pp. 12–15.
4. P. V. Rangan and H. M. Vin, "Designing File Systems for Digital Video and Audio," *Proceedings of the ACM 13th Symposium on Operating Systems Principles*, October 1991, Pacific Grove, CA, pp. 81–94.
5. H. M. Vin and P. V. Rangan, "Designing a Multiuser HDTV Storage Server," *IEEE J. Selected Areas in Commun.* **11,** No. 1, 153–164 (January 1993).
6. D. Kandlur, M. S. Chen, and Z. Y. Shae, "Design of a Multimedia Storage Server," *High Speed Networking and Multimedia Computing, Proc. SPIE* **2188,** 164–178 (1994).
7. N. Reddy and J. Wyllie, "Disk Scheduling in a Multimedia I/O System," *Proceedings of ACM Multimedia '93*, Anaheim, CA, August 1993, pp. 225–233.
8. P. Yu, M. Chen, and D. Kandlur, "Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management," *Proceedings of the Third International Workshop on Network and Operating Systems Support for Digital Audio and Video*, La Jolla, CA, November 12–13, 1992, pp. 44–55.
9. F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID—A Disk Array Management System for Video Files," *Proceedings of ACM Multimedia '93*, 1993, pp. 393–400.
10. D. N. Serpanos and T. Bouloutas, "Centralized Versus Distributed Multimedia Servers," *Research Report RC-20411*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1996.
11. D. Gemmel, H. Vin, D. Kandlur, P. Rangan, and L. Rowe, "Multimedia Storage Servers: A Tutorial," *Computer* **28,** No. 5, 40–49 (May 1995).
12. M. M. Buddhikot, G. M. Parulkar, and J. R. Cox, "Design of a Large Scale Multimedia Storage Server," *Computer Networks & ISDN Syst.* **27,** No. 3, 503–517 (December 1994).
13. A. L. Drapeau, K. W. Shirriff, J. H. Hartman, E. L. Miller, S. Seshan, R. H. Katz, K. Lutz, D. A. Patterson, E. K. Lee, P. M. Chen, and G. A. Gibson, "RAID II: A High Bandwidth Network File Server," *Proceedings of the 21st International Symposium on Computer Architecture*, Chicago, April 1994, pp. 234–244.
14. P. Lougher and D. Shepherd, "The Design of a Storage Server for Continuous Media," *Computer J.* **36,** No. 1, 32–42 (1993).
15. E. Chang and A. Zakhor, "Variable Bit Rate MPEG Video Storage on Parallel Disk Arrays," *Proceedings of the 1st IEEE International Workshop on Community Networking*, San Francisco, July 13–14, 1994, pp. 127–137.
16. A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic Batching Policies for an On-Demand Video Server," *ACM Multimedia Syst.* **4,** No. 3, 112–121 (June 1996).
17. D. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," *Commun. ACM* **34,** No. 4, 46–58 (April 1991).
18. C. Partridge, *Gigabit Networking*, Addison-Wesley Publishing Co., Reading, MA, ISBN 0-201-56333-9, 1994.
19. M. Kumar, J. L. Kouloheris, M. J. McHugh, and S. Kasera, "A High Performance Video Server for Broadband Network Environment," *Proc. SPIE* **2667,** 410–421 (1996).
20. J. Kay and J. Pasquale, "The Importance of Non-Data Touching Processing Overheads in TCP/IP," *Proceedings of ACM SIGCOMM*, San Francisco, September 1993, pp. 259–268.
21. J. Kay and J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," *Proceedings of the Winter 1993 USENIX Conference*, San Diego, January 1993, pp. 249–258.
22. R. L. Haskin and F. L. Schmuck, "The Tiger Shark File System," *Proceedings of Spring COMPCON '96*, Santa Clara, CA, 1996, pp. 226–231.

Received November 4, 1996; accepted for publication March 27, 1997

Manoj Kumar *IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (mkumar@watson.ibm.com).* Dr. Kumar received the B.Tech. degree from the Indian Institute of Technology, Kanpur, India, in 1979, and the M.S. and Ph.D. degrees in electrical engineering from Rice University, Houston, Texas, in 1981 and 1984, respectively. He is currently the manager of the Electronic Commerce Systems group. His research interests are in the use of the Internet for business activities. He is currently involved in research projects related to business negotiations on the Intenet, use of the Internet for distributing coupons and other sales promotions, and customization of a business application to accommodate the tastes and preferences of business partners (shoppers). Previously, he was involved with the architecture and design of multimedia-application servers and architecture, design, and prototyping of massively parallel computers.