Kernel Support for Live Digital Audio and Video*

Kevin Jeffay   Donald L. Stone   F. Donelson Smith

University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC, USA 27599-3175
{jeffay,stone,smithfd}@cs.unc.edu

Abstract: We have developed a real-time operating system kernel which has been used to support the transmission and reception of streams of live digital audio and video in real-time as part of a workstation-based conferencing application. An experimental environment consisting of a number of workstations interconnected with a 16 Mbit token ring has been created and used to evaluate quantitatively the performance of the kernel and conferencing application, as well as the quality of the conferences they are capable of supporting. Our early experiences with these systems are described.

Introduction

Recent advances in video compression algorithms — and their realization in silicon — have made it possible to consider introducing streams of digitized audio and video into the processing workload of workstation operating systems. For example, by outfitting workstations with off-the-shelf video cameras, microphones, digital video and audio acquisition and compression hardware, and audio amplifiers, it is possible to construct multimedia applications such as integrated voice/video/text documents and browsers [Hopper 90] as well as communication utilities such as workstation-based video and/or audio conferences [Terry & Swinehart 88, Jeffay & Smith 91].

While the hardware for such systems is readily available, existing operating systems and network communication protocols are inadequate for supporting multimedia applications such as browsing a video document or conferencing. This is due to the real-time processing requirements of digital audio and video, specifically, rigid throughput and latency requirements. For browsing or conferencing in a distributed system, frames of video must be acquired at a remote workstation (either from a camera or from a disk file) and transmitted so as to arrive at the local workstation and be displayed at the (precise) rate of one frame every 33 ms. Problems such as these have stimulated programs of basic research in many of the traditional areas of operating systems such as file systems [Rangan & Vin 91], and scheduling and inter-process communication [Anderson *et al.* 90, Govindan & Anderson 91, Jeffay *et al.* 91].

* This work supported in parts by grants from the National Science Foundation (numbers CCR-9110938 and ICI-9015443), and by the Digital Equipment Corporation and the IBM Corporation.

Our interest lies in the development of operating system infrastructure for the processing of *live* digital audio and video, specifically, workstation-based conferencing. Applications requiring live digital audio and video, such as conferencing, are unique in that their real-time throughput and latency requirements are particularly demanding. A conferencing application fundamentally requires that the audio and video data be processed *as it is generated* (*i.e.*, with zero or one buffer). To do otherwise implies that either portions of the conference will not be reproduced (*e.g.*, frames will be dropped) or that artificial latency is imposed between acquisition and display processes. In order for a system to be usable as a conferencing tool, we should minimize, if not avoid altogether, the occurrence of these events. Ideally a workstation-based conferencing system should be indistinguishable from the more traditional analog (*i.e.*, non-computer based) system.

We have constructed an experimental network of workstations capable of processing live digital audio and video and are using this system to experiment with operating system and network support for continuous-time media. In this note we describe some early experiences with a prototype conferencing application constructed on top of an operating system kernel we have built. The kernel provides real-time computation and communication services that enable a programmer to both specify real-time throughout requirements and to assess end-to-end latencies. Moreover, the kernel supports a tasking model for which it is possible to determine *a priori* if sufficient processing resources are currently available to meet an application's requirements.

In designing the conferencing application, our approach has been to view the problem as one of real-time resource allocation and control. We hope to demonstrate by example that much of the technology developed within the real-time systems community is directly applicable to the problem of supporting applications that manipulate digital audio and video. Although we have chosen to focus on live digital audio and video, we believe the infrastructure we have developed is applicable to applications that manipulate both live and recorded media.

The following section briefly describes the operating system kernel used to support real-time digital audio and video conferencing. We follow with a description of the architecture of the conferencing prototype itself and describe our initial experiences using this system. We conclude with a brief discussion of the appropriateness of the performance guarantees provided by the kernel and assess how well the kernel's programming model responds to the real-time processing needs of the conferencing application.

Kernel Overview

Operating system support for our conferencing application is provided by an operating system kernel we have developed called YARTOS (Yet Another Real-Time Operating System) [Jeffay *et al.* 91]. This kernel was developed to experiment with a paradigm of process interaction called the *real-time producer/consumer (RTP/C) paradigm* [Jeffay 89]. The RTP/C paradigm defines a semantics of inter-process communication that provides a framework for reasoning about the real-time behavior of programs. This semantics is realized through an application of some recent results in the theory of deterministic scheduling and resource allocation. We believe YARTOS to be a "general purpose" real-time operating system kernel. In addition to the conferencing application, YARTOS prototypes have been used in a 3-dimensional interactive graphics system used for research in *virtual realities*, and a HiPPI data link controller.

The programming model supported by YARTOS is an extension of Wirth's discipline of real-time programming [Wirth 77]. In essence it is a message passing system with a semantics of inter-process communication that specifies the real-time response that an operating system must provide to a message receiver. This allows us to assert an upper bound on the time to receipt and processing of each message. The exact response time requirement is a function of such factors as the rate with which a process receives messages on a given input channel. Ultimately, these rates are functions of the rates at which data arrives from external sources. These semantics provide a framework both for expressing processor-time-dependent computations and for reasoning about the real-time behavior of programs. The programming model is described in greater detail elsewhere [Jeffay 89].

YARTOS itself supports two basic abstractions: *tasks* and *resources*. A task is an independent thread of control (*i.e.*, a sequential program) that is invoked in response to the occurrence of an event. An event is a stimulus that may be generated by processes external to the system (*e.g.*, an interrupt from a device) or by processes internal to the system (*e.g.*, the arrival of a message). We assume events are generated repeatedly with a (non-zero) lower bound on the duration between consecutive occurrences of the same event. Each invocation of a task must complete execution before a well-defined deadline. The invocation intervals and deadlines for a task are derived from constructs in the higher-level programming model. During the course of execution, a task may require access to some number of resources. A resource is a software object (*e.g.*, an abstract data type) that encapsulates shared data and exports a set of procedures for accessing and manipulating the data. Like a monitor, resources guarantee mutually exclusive access to the data they encapsulate. Resources are accessed indirectly through the kernel. Support for resources is included to ensure *priority inversions* do not occur — a phenomena in which low priority processes exclude high-priority processes from accessing time-critical data, thus causing the high priority processes to miss deadlines [Sha *et al.* 90].

For a given workload (a set of tasks and resources), the goal of YARTOS is to guarantee that (1) all requests of all tasks will complete execution before their deadlines and (2) no shared resource is accessed simultaneously by more than one task. We have developed an optimal (preemptive) algorithm for sequencing such tasks on a single processor [Jeffay 90]. The algorithm is optimal in the sense that it can provide the two guarantees whenever it is possible to do so. Moreover, an efficient algorithm has been developed for determining if a workload can be guaranteed a correct execution. This algorithm forms the basis for a resource reservation protocol that is executed prior to the start of a video conference by workstations participating in the conference. In addition to its academic value, the optimality of the YARTOS resource allocation policies are important for effectively trading-off processing requirements for guaranteed response time. If YARTOS cannot guarantee a correct execution to a process, feedback can be provided on why the guarantee is not possible. A programmer can typically achieve a compromise guarantee by either relaxing the response time constraint or by improving the execution time of one or two specified processes. The optimality properties ensure that the reasons for the lack of a guarantee are fundamental in nature.

Workstation-Based Conferencing

Motivation

Our emphasis on workstation-based conferencing arises from an interest in using computers and communication networks to facilitate collaboration among scientific and technical professionals [Smith *et al.* 90]. From a technological standpoint, the goal is to support multiple, concurrent streams of digital audio and video in a distributed network of computer workstations. These streams may either form disjoint conferences within the network or involve one workstation in multiple simultaneous conferences.

While the requirements for media in conferencing systems could be met using a combination of conventional digital and analog (audio/video) technology (*e.g.*, a workstation with an adapter for analog video such as the Parallax card), by manipulating audio and video in an entirely digital format we leverage *existing* communications infrastructure (*e.g.*, local-area networks) to construct new and powerful tools for collaboration with remote colleagues. Moreover, digital formats admit the possibility of writing software to implement functions that now require specialized hardware in teleconferencing systems (*e.g.*, voice-activated controls to put the current speaker's image in a window; multi-image windows "quad-split" so that up to four participants are simultaneously visible).

Experimental Set-Up

We have built a private network to experiment with live digital audio and video. Currently we have a small number of IBM PS/2 workstations (Intel 80386 processor) interconnected with a 16 Mbit token ring network. We use IBM-Intel ActionMedia 750 adapters (based on Intel's Digital Video Interactive (DVI) technology) for the acquisition, compression, decompression, and display of digital audio and video [Luther 90].[1] The compression component of the ActionMedia system can be programmed to perform any of a number of compression algorithms that trade-off execution time for image size and quality. We use an algorithm capable of supporting real-time (*i.e.*, full-motion — 30 frames per second) compression or decompression of full color images at a resolution of 256×240 but with considerable loss of image quality. Newer high-performance versions of the display and image processors should provide larger and better quality images at 30 frames per second [Harney *et al.* 91].

Each workstation is configured with a set of ActionMedia adapters and is connected to a video camera, microphone, audio amplifier, and video monitor. Two ActionMedia adapters are required to acquire digital audio and video: (1) a capture adapter that digitizes RGB signals from a video camera along with two channels of analog audio inputs, and (2) a delivery adapter that provides (along with many other functions) capabilities for video compression, decompression, display control, and audio signal processing. The capture adapter alone is required for playback of an audio and video stream. With a pair of ActionMedia adapters, a given workstation may be either the originator or receiver of a audio/video stream, but cannot be both concurrently. With the addition of a second delivery adapter, a workstation can transmit and receive video streams simultaneously. In the following we consider only a single unidirectional audio/video stream between two workstations.

In our experiments we have used a frame-independent compression algorithm (*i.e.*, one in which the compression of a frame is not influenced by the compression of previous frames). In our present configuration, the resulting bandwidth requirement for transmitting a stream of compressed audio and video over a local area network is approximately 2 Mbits per second. A compressed audio frame is approximately 0.5 Kbytes and a compressed video frame is, depending on the scene, approximately 6.5-7.5 Kbytes.

In addition to the digital system, we have also constructed a separate analog video conferencing system using an existing in-house CATV system. The output from the workstation camera simultaneously supplies the digital and analog systems with video input thereby providing a convenient mechanism for assessing the (qualitative) latency and image quality of the digital system. A preliminary comparison of the two systems is reported below.

[1] ActionMedia are Digital Video Interactive are registered trademarks of the Intel Corporation.

Software Architecture

The conferencing application runs as a user program on top of the YARTOS kernel. There are separate applications for originating and receiving a conference. Figure 1 shows design of the conference origination application. The conference reception application is similar. The origination application is responsible for acquiring, compressing, and transmitting a video stream. As shown in Figure 1, the application consists of 5 tasks (represented by circles) and 2 resources (represented as a collection of shaded circles). In Figure 1 single headed arrows indicate message channels. In YARTOS these provide control flow information. Double headed arrows indicate global data flow. Omitted from Figure 1 are the kernel-level interrupt handlers.

The origination application is controlled by two externally generated signals. The network adapter signals an interrupt for one of two events. The *Transmit Ready* interrupt, TR, is generated sometime after an application initiates a network transmit. This signals that the network adapter is ready for the data. The *Transmit Complete* interrupt, TC, is generated when a packet has been transmitted. The ActionMedia hardware signals an interrupt for one of three events. Two *Vertical Blanking Interrupts*, VBI0 and VBI1, are generated after each half of a frame of video has been scanned. A *Compression Complete* interrupt, CC, is generated after a frame of video has been compressed.

Each video frame goes through a three stage pipeline: digitization, compression of the digitized image, and transmission of the compressed data over the network. At any given time, there are three frames in the pipeline. The pipeline is initiated by a VBI1 signal. A digitize is initiated ("scheduled") by providing an address of a buffer to the capture adapter. Two VBI0s later (the first VBI0 signals that the digitize has begun — the second VBI0 signals that it is complete) the frame has been digitized. After the second VBI0, the compression is initiated. As the frame is compressed, the hardware reads digitized video from one buffer and deposits compressed video data into another. The CC interrupt signals the end of this operation. At this point, the transmission over the network is initiated. Lastly, when the network adapter is ready to transmit, it raises the TR interrupt and network packets are transferred onto the adapter. The results of a stage in the pipeline are communicated to later stages through two buffer pools (two YARTOS resources) that are shared between software tasks (as shown in Figure 1) and hardware tasks (as described in Table 1). One buffer pool holds digitized frames, the other holds compressed frames.

The timing of the execution of all tasks is critical to the functioning of the pipeline. The response time guarantees provided by the YARTOS kernel are used to ensure correct operation of the pipeline. For example, since a digitize always begins at a VBI0, the Schedule Digitize task (invoked by VBI1 messages) must be completed before the next VBI0. VBI interrupts occur at intervals of 16.5 ms, so this task must complete within 16.5 ms. of its invocation. Table 1 summarizes the sequence of hardware and software operations required to acquire, compress, and transmit a frame of video and specifies the response time requirements of the software tasks and the assumed hardware timing characteristics.

Experimental Results

We have run a number of experiments with unidirectional conferences (one origination application and one reception application) to test both the performance of our system and the qualitative nature of the conference. (For unidirectional digital conferences, we have the capability of providing audio and video channels in the reverse direction with our analog system.) The measures of interest are the intra-workstation and intra-network latency for a frame of video and the effect of dropped or missed frames on the perceived quality of the conference.

Table 1: Hardware/Software pipeline for transmitting a video frame.

Operation	HW/SW	Function	Timing Properties
Schedule Digitize	SW	Get a free digitize buffer a and initiate the digitization of the next frame of video.	Must complete within 16.5 ms.
Digitize	HW	Scan a frame of video into a digitizing buffer.	Completed in 33 ms.
Schedule Compress	SW	Get a digitizing buffer and initiate the compression of the most recently digitized video frame.	Must complete within 16.5 ms.
Compression	HW	Compress a frame of video from a compress source buffer into a compress sink buffer.	Completed in 20 ms.
Transmit	SW	Get a compress sink buffer, make it into a network packet, and initiate a network transmit.	Must complete within 20 ms.
Initiate Transmit	HW	Signals that a buffer is available on the token ring adapter.	Completed in less than 1 ms.
Network Driver	SW	Copy data from a compress buffer onto the token ring adapter.	Must complete within 16.5 ms.
Transmission	HW	Physically transmit data.	A function of network load.

For the conference origination application we define the intra-workstation latency for a frame of video as the difference between the time a network packet containing the data for a frame is transmitted and the time the digitization of the frame started. We can demonstrate analytically that the worst case latency is 3.5 frame times (approximately 116 ms.) for this application. (This assumes an otherwise idle network.) The conference reception application is structured similarly and has similar latency. Figure 2 illustrates the worst case interleaving of the software and hardware operations for the origination application. In Figure 2, hardware processes execute throughout the intervals shown for hardware processes. Software tasks execute somewhere between the left and right endpoints of the intervals shown for software tasks.

In practice we estimate the end-to-end latency in a conference to be between 6 to 7 frame times (200 ms. - 230 ms.). This latency in the digital system is easily noticeable when compared side-by-side with the analog system. However, when the system is used for conferencing, specifically, when users are physically separated and there there is no reference standard, we have found the digital system to be adequate. We conjecture that this is because in conferencing there is little physical movement in front of the camera and often only one individual speaks at a time. We further conjecture that without synchronizing the sending camera and the receiving display, it is not possible to achieve a worst case end-to-end latency of less than 5 frame times in the present generation ActionMedia system. (A latency of 5 frame times would require an infinitely fast processor and network.)

Although we are limited in our present experimental configuration to only originating or receiving 30 frames of video per second,[2] we are confident that the latency guarantees provided by YARTOS for a single video stream would remain unchanged as the number of video streams manipulated by a workstation increases — provided that the processor does not

[2] We simulate multi-person conferences by displaying video images at a receiving station at a reduced rate (*e.g.* 15 frames per second for displaying two remote conferees) and replaying the audio from only one stream at a time. (Two streams of audio can be played simultaneously by playing one channel from each stream.)

become saturated. If a task with a minimum inter-invocation time of p time units is guaranteed a response time of p time units, then YARTOS is effectively reserving c/p of the processor, where c is the cost of executing the task, for the execution of this task. So long as YARTOS does not over commit the processor, the task will be guaranteed a response time of p time units independent of the number and processing requirements of other tasks in the system.

We have run a number of experiments on our network to determine how the quality of a conference degrades as a function of the delay in the network. In the initial experiments reported here we use do not use any of the priority reservation mechanisms of the token ring. Conference quality is assessed in terms of the number of "frame incidents" observed in an interval. A frame incident occurs at the conference receiver when a previously played frame has to be replayed. This occurs when either a frame is discarded by the origination application (because the network has not transmitted frame n by the time frame n+1 is ready for transmission), or when a frame arrives "late" at the receiver.

In assessing the quality of a conference, it should be noted that for our use of the ActionMedia system, faithful reproduction of the audio component is paramount. This is because audio data is acquired in fairly large blocks (33 ms. worth). In the conferencing application, an observer can (easily) detect a single dropped or replayed audio frame whereas several consecutive video frames need to be dropped in order for a user to notice. Since we currently transmit audio and video frames in the same network packet it is not possible to drop/replay one frame without doing the same to the other. Therefore while our emphasis is primarily on audio frames, we will speak only of dropped/replayed frames.

In the current implementation of a unidirectional conference, there is no explicit synchronization between the sending and receiving workstations. (To add synchronization would fundamentally add latency.) The origination workstation transmits frames at an aggregate rate of 1 frame every 33 ms. (with a measured jitter of ±2 ms.). The display tasks on the receiving workstation side are driven off (local) VBI interrupts. On each VBI0, a frame from a queue (typically of length one) of received frames is inserted into a decompression/display pipeline. If the receiving side is ahead of the sending side, *i.e.* a frame arrives late, then the receiver replays the previous frame. Such an event typically occurs at most once (and typically within the first few frames) for a light to moderately loaded network.

Our preliminary observations indicate that a frame incident rate of more than 2-4 incidents per 1000 frames is noticeable (*i.e.*, annoying).[3] Moreover, we observe that as the delay in the network (in our case due to artificially generated traffic) as seen by a workstation originating a conference, approaches 16 milliseconds, the occurrence of frame incidents increases dramatically and hence the audio quality and the quality of the conference itself deteriorates rapidly. This can be explained by noting that some of the latency in the conference is due to the fact that the (hardware) video processes generating VBI interrupts on the originating and receiving machines are not synchronized. If there were no delay in the network then one would expect that on average a frame of audio and video would be queued for 16 ms. (*i.e.*, half a frame time) at the receiving workstation before entering the decompression/display pipeline. Therefore, on average, the receiving workstation should be immune to substantial delays in the network. (For a 16 Mbit token ring, a 10 ms. delay would correspond to a utilization of the network bandwidth of approximately 75% [Bux 89].)

[3] These experiments were performed using a CD player as the audio input source on the originating machine. It is not clear how the threshold on frame incidents would differ for voice. We plan to perform additional experiments with live voice input.

On Performance Guarantees

The notion of a response time or other performance guarantee is central to our work and is indeed essential for supporting applications that manipulate live digital audio and video. As one example, if the Schedule Digitize task does not complete execution within 16 ms. of receiving a VBI1 message, then a frame of video and audio necessarily will be lost. Through careful attention to processor scheduling, YARTOS provides the desired guarantees. Moreover, we can demonstrate both analytically and empirically that frames are not dropped by a workstation originating a conference because of the workload in the system. There are, however, two aspects of these performance guarantees that need to be examined in closer detail to determine how well YARTOS supports the real-time requirements of the conferencing application.

The first concerns the usefulness of the guarantees. For a given programming model that includes repetitive real-time processing constraints, it is typically not very difficult to derive sufficient conditions on the operating environment — called *schedulability* conditions — that will ensure that the real-time response properties of tasks will be met. The more interesting question is how accurate these conditions are. That is, if a set of tasks do not satisfy the conditions then does this necessarily imply that a deadline will be missed if the tasks are executed? Moreover, if this is indeed the case then how can the programmer ameliorate this situation? For YARTOS, it is the case that if a set of tasks do not satisfy the schedulability conditions then it is possible to demonstrate a maximal, finite set of orderings of events (*e.g.*, message arrivals) that will necessarily lead to a missed deadline. If the programmer is willing to certify that none of these sequences of events can happen then no deadlines will ever be missed.

The second, and likely more important issue, concerns the fit between the programming model exported by YARTOS and the processing needs (both real-time and non-real-time) of the conferencing application. Although YARTOS can provide meaningful real-time guarantees, these guarantees come at a cost of a fairly restrictive programming model. YARTOS was designed to to support applications whose real-time processing constraints arise from the need to process data at a precise rate. Therefore, each task has a notion of a minimum inter-arrival time of activation messages and a deadline based on this inter-arrival time. This meshes nicely with the processing required to respond appropriately to VBI interrupts — *i.e.*, it is periodic and has a well-defined deadline. It is not well suited, however, to support some of the processing required to respond to a CC interrupt. This is because there are several logical operations that are executed in response to a CC interrupt but not all of these operations have the same deadline. In particular, some of these deadlines are not a function of the inter-arrival time of the CC interrupt. One important function has been omitted from Figure 1: the movement of digitize and compress buffers from the compress source and transmit queues respectively, back to their corresponding free queues. In the case of digitize buffers on the compress source queue, at the occurrence of a CC interrupt, a digitize buffer on the compress source queue may be moved to the free queue.

A careful analysis of Figure 2 reveals that the conference origination application can, in principle, work with three digitize buffers. This can be seen by noting that at the fourth VBI0 in Figure 2 (the start of the digitization of the unshown fourth frame), the hardware compression of frame 1 must have completed and hence the digitize buffer used for frame 1 can be reused for frame 4. If, however, the operation to free the digitize buffer is performed as part of the Transmit task, it is possible that the free operation for the digitize buffer used for frame *n* may not take place before execution of the Schedule Digitize task for frame *n*+3. This is because under YARTOS the deadline for the Transmit task can be expressed only in terms of the when the next CC message is expected to arrive. In general, this deadline is insufficient for ensuring that a digitize buffer can be freed before the next invocation of the Schedule

Digitize task (*i.e.*, before the next VBI1). Therefore, if the buffer free operation is performed by the Transmit task then the implementation of the digitize buffer resource must provide four buffers.

Abstractly, this is solely an issue of efficient resource utilization since the addition of a fourth digitize buffer would have no impact on the real-time performance of the application. In the current configuration of our system, however, this is a critical issue as digitize and compress buffer resources are implemented in memory on the ActionMedia adapter and there is insufficient space for four digitize buffers.

The root of the problem is that the true deadline of the buffer free operation is not a function of the interarrival time of CC interrupts and hence cannot be directly supported with a YARTOS task. The solution we have adopted is to associate an *eventcount* [Reed & Kanodia 79] with each message port in the system. Whenever a message is sent to a task the eventcount is incremented. The eventcount is then used for producer/consumer synchronization on events as described in [Reed & Kanodia 79]. In the case of the digitize buffer-free operation, this operation can now be performed when it is needed as a side effect of the resource call to remove a free digitize buffer. When this call is made, the kernel is invoked to check the eventcount for the Transmit task. The value of this event count indicates the number of CC interrupts that have occurred and this corresponds to the number of times digitize buffers have been freed. If the CC eventcount differs from the VBI1 eventcount (indicating the number of times free digitize buffers have been removed) by less than the number of digitize buffers in the system there is indeed a free digitize buffer.

In general, the eventcount mechanism must be a kernel provided function because the recognition of the event and the increment of the eventcount must be indivisible in order to preserve the semantics of eventcounts.

Future Directions

In the near term we will be experimenting with alternate conference application designs and measuring their performance. The goal is to characterize the cost of an "acceptable" quality conference in terms of the observed rate of frame incidents, the latencies induced by (a generic model of) the video hardware, the operating system and the network, and to demonstrate how fluctuations in the values of one component affect the others and the conference itself.

With regard to the conferencing application itself, of particular interest is the separation of audio and video into independent network packets. In the ActionMedia system, compressed audio data for a frame is available 33 ms. before the corresponding compressed video data and hence could be transmitted significantly earlier than the video data. Given that the quality of the audio in a conference is the primary indicator of overall conference quality, by transmitting audio as soon as it is available we should be able to tolerate substantial network delays (*e.g.*, delays on the order of 80 ms.). This is, of course, an artifact of the 200 ms. end-to-end latency for video data in our system.

Given that our emphasis on workstation-based conferencing arises from an interest in using such systems to facilitate collaboration among scientific and technical professionals, we are striving to integrate our kernel and conferencing applications into a computing environment that includes UNIX workstations. To this end, we are currently adding ethernet support and porting a TCP/IP implementation to the YARTOS kernel. The token ring network (and hopefully its FDDI successor) will remain primarily a private network for experiments with real-time communications protocols. In addition we are working on porting an X server to YARTOS. In essence we hope to construct a real-time multimedia workstation that provides a window onto existing computing environments while providing new, real-time communications and continuous media services.

Summary and Conclusions

The YARTOS programming model provides response time guarantees to tasks based on their minimum inter-invocation time. While this basic mechanism has not supported the conferencing application seamlessly, it has been sufficient to construct the system. In particular, the accuracy of the YARTOS schedulability analysis has been most useful as it has allowed us to concentrate on issues of logical correctness while ignoring efficiency considerations. (For example, we have constructed, and received the desired performance guarantees, for conferencing applications that utilize close to 80% of the CPU.)

Concerning our experiments, we have empirically determined that when the delay in the network exceeds 16 ms. then the perceived quality of the conference falls off sharply (more than 4 dropped/replayed frames per 1000). Therefore, for our system we posit that if P(*network delay* < 16 ms.) > .996, no reservation or priority mechanisms are required to ensure good fidelity conferences on our (token ring) network.

References

Anderson, D.P., Tzou, S.-Y., Wahbe, R., Govindan, R., Andrews, M., 1990. *Support for Continuous Media in the DASH System*, Proc. Tenth Intl. Conf. on Distributed Computing Systems, Paris, France, May 1990, pp. 54-61.

Bux, W., 1989. *Token-Ring Local-Area Networks and Their Performance*, Proc. of the IEEE, Vol. 77, No. 2, (August), pp. 238-256.

Govindan, R., Anderson, D.P., 1991. *Scheduling and IPC Mechanisms for Continuous Media*, Proc. ACM Symp. on Operating Systems Principles, ACM Operating Systems Review, Vol. 25, No. 5, (October), pp. 68-80.

Harney, K., Keith, M., Lavelle, G., Ryan, L.D., Stark, D.J., 1991. *The i750 Video Processor: A Total Multimedia Solution*, Comm. of the ACM, Vol. 34, No. 4 (April), pp. 64-79.

Hopper, A., 1990. *Pandora — An Experimental System For Multimedia Application*, ACM Operating Systems Review, Vol. 24, No. 2, (April), pp. 19-34.

Jeffay, K., 1989. *The Real-Time Producer/Consumer Paradigm: Towards Verifiable Real-Time Computations*, Ph.D. Thesis, University of Washington, Department of Computer Science, Technical Report #89-09-15.

Jeffay, K., 1990. *Scheduling Sporadic Tasks With Shared Resources in Hard-Real-Time Systems,* University of North Carolina at Chapel Hill, Department of Computer Science, Technical Report TR90-038, August 1990. (In submission.)

Jeffay, K., Smith, F.D., 1991. *System Design for Workstation-Based Conferencing With Digital Audio and Video*, Proc. IEEE Conference on Communication Software: Communications for Distributed Applications and Systems, Chapel Hill, NC, April 1991, pp.169-178.

Jeffay, K., Stone, D., Poirier, D., 1991. *YARTOS: Kernel support for efficient, predictable real-time systems*, to appear: Proc. IFAC Workshop on Real-Time Programming, Pergamon Press.

Lederberg, J., Uncapher, K., (eds.), 1989. *Towards a National Collaboratory: Report of an Invitational Workshop at the Rockefeller University*, March, 1989. Distributed by the National Science Foundation.

Luther, A.C., 1990. "Digital Video in the PC Environment," McGraw-Hill, Second Ed.

Rangan, P.V., Vin, H.M., 1991. *Designing File Systems for Digital Video and Audio*, Proc. ACM Symp. on Operating Systems Principles, ACM Operating Systems Review, Vol. 25, No. 5, (October), pp. 81-94.

Reed, D.P., Kanodia, R.K., 1979. *Synchronization with eventcounts and sequencers*, Comm. of the ACM, Vol. 22, No. 2, (February), pp. 115-123.

Sha, L., Rajkumar, R., Lehoczky, J.P., 1990. *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*, IEEE Trans. on Computers, Vol. 39, No. 9, (September), pp. 1175-1185.

Smith, J.B., Smith, F.D., Calingaert, P., Hayes, J.R., Holland, D., Jeffay, K., Lansman, L., 1990. *UNC Collaboratory Project: Overview*, University of North Carolina at Chapel Hill, Department of Computer Science, Technical Report TR90-042.

Terry, D.B., Swinehart, D.C., 1988. *Managing Stored Voice in the Etherphone System*, ACM Trans. on Computer Systems, Vol. 6, No. 1, (February), pp. 3-27.

Wirth, N., 1977. *Toward a discipline of real-time programming*, Comm. of the ACM, Vol. 20, No. 8 (August), 577-583.

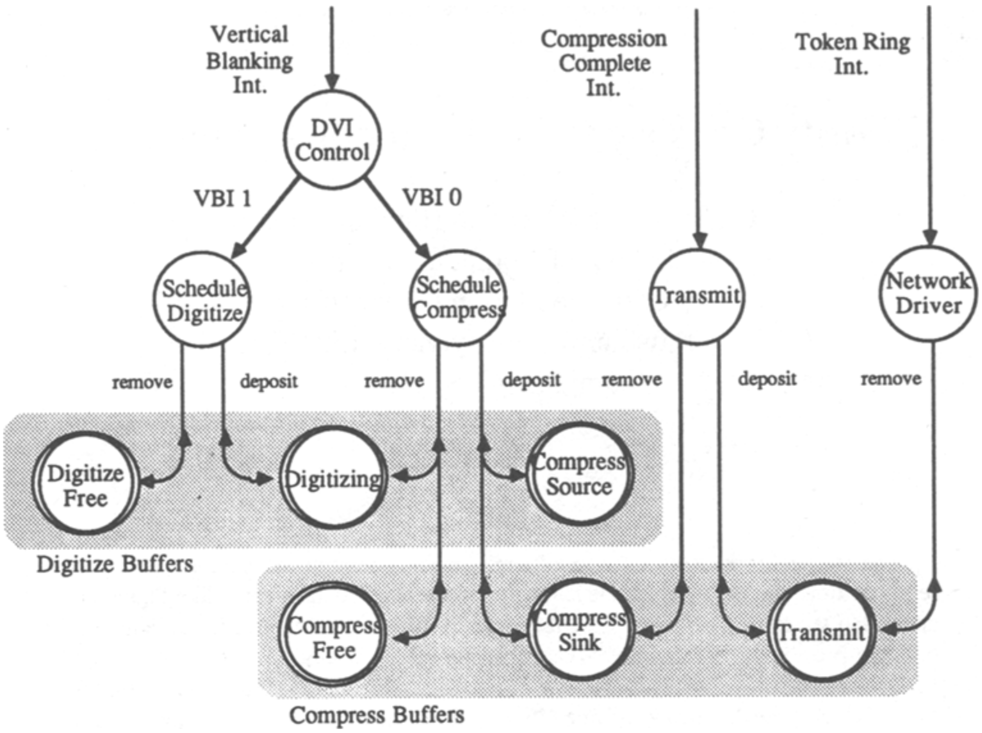

Figure 1: Conference origination application.

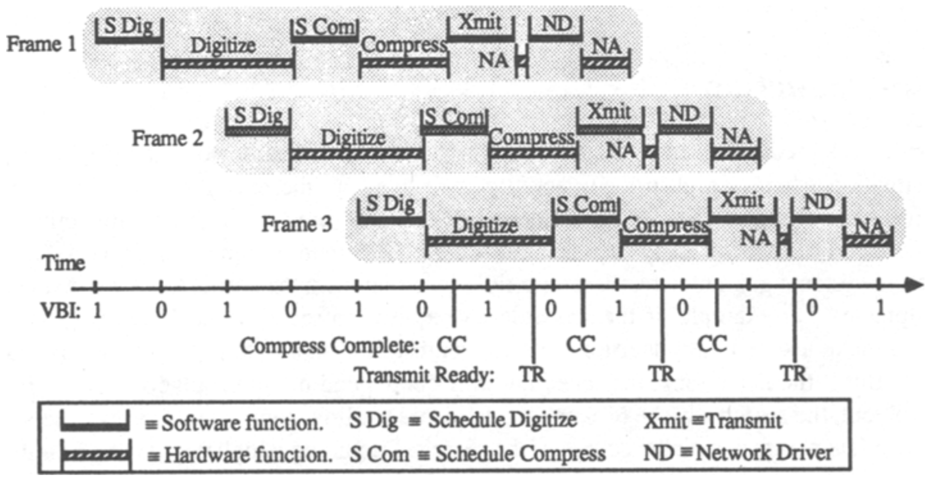

Figure 2: Execution of the conference origination application for the processing of three frames of video.