An Evaluation of Priority Consistency in Protocol Architectures

Clifford W. Mercer and Hideyuki Tokuda
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Workstations for digital audio and video applications must support predictable processing, presentation, and transmission of continuous media data while making efficient use of limited hardware resources. We present techniques for managing network resources and processing messages in a predictable fashion. Our solutions use a novel structure for the protocol processing software, and we examine several alternatives and discuss the engineering tradeoffs associated with each. Using simulation, we evaluate our protocol processing structures and show that the more sophisticated structures can improve the response time and predictability of time-constrained messages. The ideas presented in this paper can be applied to resource management in the operating system and to the structuring of application-level servers. The designers of gateway machines (for internetwork routing) which handle both time-constrained traffic and non-time-constrained traffic may also benefit from this study.

1 Introduction

The current trend in workstations is toward computing and communication stations capable of supporting traditional applications along with video phone, video conferencing, and other continuous media applications. Several recent entries in the workstation market have digital audio and graphics capability and represent our first steps into this new technology. But the additional complexity of the hardware and the timing requirements of digital audio and video applications present difficult problems for the operating system designer to resolve.

This research was supported in part by the U.S. Naval Ocean Systems Center under contract number N66001-87-C-0155, by the Office of Naval Research under contract number N00014-84-K-0734, by the Federal Systems Division of IBM Corporation under University Agreement YA-278067, and by the SONY Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of NOSC, ONR, IBM, SONY, or the U.S. Government.

Continuous media applications require special support from the operating system and network, and even though some work has been done on abstractions for continuous media [3, 17] and support for resource reservation [2], very few results have appeared which suggest how these abstractions and reservation systems are to be effectively implemented. Typical proposals suggest assigning priorities or using deadline information for scheduling. These ideas are definitely useful for continuous media systems, but unfortunately, assigning priorities does not guarantee that the activities will actually execute in the desired order. Because continuous media applications may need to synchronize or communicate with each other, *priority inversions* [20] may occur where a high priority activity is delayed by a lower priority activity, and thus the *priority integrity* of the system is compromised. It is not possible to completely eliminate priority inversions in a system, but it is possible, using sophisticated scheduling algorithms, synchronization primitives, and communications structures, to reduce the effects of priority inversion while keeping overhead to a minimum.

In this paper, we concentrate on the performance of the protocol handling software or protocol engine of a distributed operating system. We use the word "performance" in the general sense, i.e. the ability of the system to achieve its objectives, rather than in the narrow sense of the ability of the system to score well on a few specific metrics. Continuous media applications will depend on the ability of the network to move digital audio and video data and the ability of the network software to schedule protocol processing and other message handling activities in a timely fashion. In the simplest case, we can divide the workload into "important" time-constrained activities and other activities like file transfer which are not time-constrained but still require reasonable service. The first step might be to assign priorities to messages: high priority for time-constrained messages and low priority for non-time-constrained messages. So at some level, a protocol header contains a priority field and the corresponding protocol uses priority queueing instead of FIFO queueing.

Several problems arise, however, in this situation. The first problem is that the protocol processing that occurs below the prioritized protocol will not be handled according to the priority of the message, especially on the receiver's side where protocol layer encapsulation prohibits the lower-level protocols from "peeking" into the headers of higher-level protocols to check the priority of the incoming message. The messages must be processed in FIFO order until the priority is revealed at the protocol layer where priorities are handled. An example of this problem is the case where there is a significant amount of processing occurring in the unprioritized layers of protocol and many low priority messages arrive followed by a high priority message. The high priority message will have to wait for each of the lower priority messages to pass through this initial processing phase, and we have a case of priority inversion. The second problem is that even if the messages are queued according to priority (above the prioritized protocol layer), priority inversion may still occur. For instance, if a low priority message is received and begins protocol processing, a high priority message may have to wait for the low priority message to complete protocol service. An alternative would be to preempt the protocol processing of the low priority message in favor of the high priority message, reducing the high priority message's delay to the duration of a context switch rather than the duration of the low priority message's protocol processing time. Another problem is that there may be a need for more than two classes of priority; perhaps four or eight priority classes are needed.

Currently, performance gains for each level of the system including processors, network devices, network driver software, and protocol engines depend on pipeline architectures and caching. The central idea of such techniques is that all of the work relating to a single activity, such as the processing associated with a message coming from the network, should be done at once so that the execution path can be streamlined. Unfortunately, this type of performance optimization requires that the execution path be non-preemptable since preempting would mean flushing the pipeline and refilling the cache. Time-constrained activities often require a high level of preemptability to assure fast response time and to reduce priority inversions, so there is a basic conflict between performance tuning techniques and response time requirements. For instance, suppose that a protocol implementation is streamlined for the normal case where a message arrives from the network device driver and passes up through several layers of protocol to the user level. Naturally, the protocol designer would make this computation non-preemptable to avoid having another activity take over the processor and flush the pipeline and change the contents of the caches. This protocol processing may take .5 to 5 milliseconds or more. Now if a low priority messages arrives from the network and enters this non-preemptable region and then a high priority message arrives, the high priority message will have to wait for the low priority message to finish executing before it can move forward.

The following section proposes several techniques for structuring the protocol processing software in an operating system. We analyze the performance of these techniques with regard to predictability (variance in the performance), efficiency, and success in meeting timing constraints. The performance results from simulations are presented for many of the techniques. Finally, we give our conclusions about the level of predictability which can be achieved using the various methods.

2 Message Scheduling Techniques

We have previously shown [9, 12, 19] that priority inversions in the protocol processing software in an operating system can adversely affect the performance of time-constrained tasks. In this study, we explore the issue in more detail and attempt to learn precisely how various approaches perform in this situation and what aspects of these techniques are most valuable. One objective is to learn how a prioritized, non-preemptable protocol engine compares with an implementation using prioritized, preemptable protocol threads. The latter is certainly more expensive in terms of overhead, and we want to determine if the additional predictability is worth the overhead.

2.1 Traditional Time-Sharing Techniques

The BSD4.3 operating system handles network protocol processing at the "software interrupt level" which has a higher priority than any process in the system [10]. The only type of processing that can take precedence over protocol processing is hardware interrupt handling. This means that low priority network activity can preempt high priority local computation, a clear case of priority inversion. We will refer to this as the SOFTINT technique. In order to introduce more sophisticated scheduling of message traffic, we use a thread-based protocol processing engine. We implement the protocol software in a single thread running at the highest priority in the system. If we use FIFO queueing for incoming messages (T1F, the single-thread FIFO queueing technique), we can compare this implementation to the software interrupt level approach to determine the overhead involved in going to a thread-based approach.

2.2 Prioritized Queueing Techniques

Using a single protocol processing thread running at high processor priority and a priority queueing discipline for incoming messages (T1P, the single-thread prioritized queueing technique), we can hopefully expect some improvement in the response time for high priority messages. So instead of a high priority message having to wait for the protocol processing of all lower priority messages that may be in the queue when it arrives, the high priority message will be delayed only by messages of higher or equal priority which arrived previously, and it will have to wait for at most one lower priority message to finish protocol processing. Now the problem with this approach is that, although high priority messages are getting good service, other high priority computations in the system which are competing with the protocol processing thread for the processor may be preempted. The system still suffers from priority inversion. For example, suppose we have a high frequency, periodic local computation to which we assign a high priority. If several low priority messages are received from the network, the protocol processing thread (running with the highest priority in the system) will run and the other high priority task will potentially miss deadlines as a result.

2.3 Multi-Threaded, Prioritized Techniques

It seems that we have to distinguish between different priority levels in the network activity so that we can schedule this activity against the other computational activities in the system. This coordination of activities across scheduling domains is an important issue in real-time systems [20]. There are two ways that we can organize the protocol processing engine (or any other server [13]) to distinguish between message priorities: one is to use the priority of the incoming message to determine the priority of the protocol thread(s) and the other is to use a pool of prioritized servers.

We consider the case where the priority of the incoming message is taken to be the priority of the thread while servicing that message (T1PI, single-thread priority inheritance technique). For now, we will ignore the problem of having more levels of process priority than message priority. This problem can be addressed using a mapping of process priority to message priority [14]. The T1PI technique has several advantages. The activity in the protocol engine associated with a particular message competes for processor cycles at the priority of the message so that no local computations are preempted in favor of low priority network traffic. Also, the high priority network traffic can effectively compete against the low priority local activities. The disadvantage is that if a low priority message is being serviced at low processor priority and a high priority messages arrives, the high priority message cannot begin processing until the low priority message is finished. But since the low priority message is running at low processor priority, any number of medium priority local activities can preempt this low priority message activity; thus priority inversion occurs. A possible solution to this problem is to have the processor priority of the thread take on the highest priority of any of the pending messages. In effect, the arrival of a high priority message causes the priority of the thread to be "bumped" to the higher priority of the new message. We call this the T1PIB technique (single-thread with priority inheritance and priority bumping).

Another way that we can reflect the message priority in the process priority of the activity which is servicing the message is to use multiple threads, each with a fixed priority. This requires that we divide the message priority levels into classes with one class corresponding to each thread. Then each message is processed at the processor priority of its class, and it does not interfere with higher priority classes and lower priority classes do not interfere with it. If the number of threads is small, the classes may have to contain many different message priority levels, and some priority inversion will result from this. If the number of threads is large, say equal to the number of message priority levels, then there is no priority inversion due to a priority granularity problem. We refer to this as the multi-threaded, prioritized technique (T2P, T4P, TnP where the number defines the number of threads and n means that the number of threads is equal to the number of processor priority levels).

An alternative to this technique is the multi-threaded, priority inheritance technique (T2PI, T4PI, TnPI). Here, the priority of the thread is inherited from the message, but there is a pool of more than one thread. The drawback of this approach is that if many low priority messages acquire threads to service them, an arriving high priority message will have to wait for a thread to become available before service can begin. An improvement would be to queue message of the same priority at the same thread, and then the threads are allocated to priority levels instead of individual messages. This addresses the drawback to some extent, but if the number of threads is smaller than the number of message priority levels, there is still some potential for lower priority traffic to consume all of the threads leaving a high priority message waiting.

These various techniques are summarized in Table 1 which contains a brief description of the distinguishing features of each technique.

3 Analysis

In order to effectively compare the proposed techniques experimentally, we must consider what we would like to know about the performance of the system and which metrics are appropriate for determining that information. Typical measures used in evaluating the performance of operating systems include throughput, average utilization, av-

Technique	Number of Threads	Processing Priority	Message Priority	Additional Comments
SOFTINT	0	Higher than any other activity	None	Preemptability very limited.
T1F	1	Higher than any other activity	None	Same as SOFTINT except for extra thread overhead.
T1P	1	Higher than any other activity	Priority queueing	Better since messages are queued according to priority, but local high priority activities may suffer.
T1PI	1	Inherited from message being processed	Priority queueing	Relieves suffering for local high priority activities, but high priority messages may get stuck.
T1PIB	1	Inherited from highest priority waiting message	Priority queueing with bumping	Avoids having high priority processes get stuck for more than the duration of protocol processing of current activity.
T2P	2	Statically assigned (high and low)	Priority queueing	Compromise on use of priority. Some priority inversion due to granularity problems.
TnP	n	Statically assigned (n levels)	Priority queueing	n is the number of processor priority levels. No granularity problem.
T2PI	2	Inherited from message being processed	Priority queueing	Problem if low priority traffic gets in before high priority traffic.
TnPI	n	Inherited from message being processed	Priority queueing	Same as T2PI.

Table 1: Summary of Techniques

erage response time, and many others. We are, of course, interested in these measures, but for the evaluation of continuous media systems, other measures are more important. For instance, the continuous media systems typically handle computations involving audio data and this activity demands a certain level of predictability from the system. Thus the variance of various measurements emerges as an important consideration. We are willing to sacrifice response time performance for low priority activities in return for better or more consistent response time performance for high priority activities.

Suppose that we have a distributed system designed to move audio data streams around. If we are sampling the audio, we will have data being generated periodically, being transferred from one machine to another periodically, possibly being filtered and then being consumed periodically by a D/A device connected to a speaker. We can easily calculate the processing requirements of this activity (assuming that the network is fairly reliable), and we can assign to this activity the highest priority in the system to try to ensure that other activities will not interfere. Now we want to consider the types of measurements to take in this system to judge whether the system is performing well for these requirements.

3.1 Basic Measures

One possible measurement is the time it takes to send and acknowledge a message containing audio samples; we call this the message latency. But we are interested not only in the mean message latency, but the maximum, minimum and variance for the latency. Another important measure is the number of deadlines missed in these periodic activities. If the message latency is so long that the task overruns its period, we say that the deadline for that period is missed, and the task will start again at the beginning of the next period (or try to complete the computation late, depending on the policy). We would like to know how often this happens. Also, a filtering activity which does not send any message on the network but just does some local computation periodically may miss deadlines in the same way, and this is a phenomenon that we want to measure as well.

3.2 Utilization-Based Measures

These are all common metrics for timesharing systems and real-time systems. However, these measures do not tell the whole story about the system. We have described how priority inversion in the system might adversely affect the performance of high priority tasks in the system, but there has been no way to quantify the degree of priority inversion in the system. We now introduce a measure of priority inversion which we call the *priority inverted utilization*. We define utilization, in the usual way, as the fraction of time during a particular interval that the processor is actively working. We define the priority inverted utilization as the fraction of time during an interval that the processor is working on a task which is not the highest priority ready task. Therefore an ideal system with perfect preemptability would never suffer from priority inversion and the priority inverted utilization would always be zero. We define the *priority inversion factor* to be the ratio of the priority inverted utilization to the total utilization for an interval.

This measure is very important for comparing systems or programming techniques. Clearly, as system with perfect preemptability would be optimal since low priority activity would be completely "invisible" to high priority activity. If the preemptability is imperfect (as with any practical system), the system will suffer some effects of priority inversion, and the low priority activities will sometimes interfere with the high priority activity leading to some level of unpredictability in the high priority tasks. A measure of the degree of this disturbance will help us to evaluate and quantify the overall predictability of the system.

4 Simulation Results

We have chosen to use simulation to evaluate the various techniques for structuring protocol processing software in an operating system. With a simulation, we can more easily test widely differing techniques and we have more control of system parameters such as protocol processing time and context switch time. In order to evaluate the techniques, we must choose a task set which is representative of a system with the types of activities that we are interested in. We have chosen a workload with audio activities since the workstation technology for this type of application is available, and the computational demands are within the capabilities of this technology. This choice is based on our experience with various real-time task sets and real-time benchmark suites.

4.1 The Task Set

In our task set, we want to measure the difference in performance between time-constrained activities with communication requirements as well as activities with no communication requirements. We need to have some background activities contending with the time-constrained processes to model the case where we have audio/video programs, editors, compilers and other activities on a single machine.

We choose audio activities which transfer audio streams from remote machines to a single audio server. In order to get two activities with different timing requirements, we take one audio task to be 16-bit audio and the other to be 8-bit audio. On the same audio server, we assume there is some kind of local activity that is filtering or generating audio signals locally and that has stringent timing requirements as well. And finally, we must consider the nature of the background activity. We want network traffic from other non-time-constrained activities to be serviced on the same machine as the audio server so that we can generate the contention in the protocol processing software. Using a background load which is constant does not really help; there is still plenty of time between the widely spaced background messages to do the time-constrained activity unless the load is very high. Of course, if the load is too high, there will be no hope of completing the audio computations. It is clear that most workstations in practice can *usually* handle the requirements for audio, but the problem comes every once in a while when a spike comes from the network or from some local computations. We are most interested in having the system behave predictably even during a spike of background activity.

In order to evaluate the performance of a technique during these spikes, we see how the system performs with more and more severe spikes. The background activity must be scalable so that we can increase the size of the spike. To do this, we use periodic background tasks which generate a message during each period, and we synchronize the periods so that messages arrive at the same time, thus creating a spike. The size of the spike is increased by increasing the number of periodic background tasks.

Based on the timing requirements of the activities, we assign priorities in a rate monotonic fashion [11] with the highest priority going to the highest frequency tasks and the lowest priority going to the background tasks. Thus we have the following tasks:

- 1 16-bit audio task at high priority (20 ms period),
- 1 8-bit audio task at medium priority (40 ms period),
- 1 high priority local computation (20 ms period) and
- n low priority tasks (100 ms period).

The value of n is a parameter of the task set which we increase to obtain a larger spike. Figure 1 illustrates the structure of the task set. The 16-bit audio stream goes from τ_1 to τ_2, the 8-bit audio stream goes from τ_3 to τ_4, τ_5 is the local high priority computation, and the background traffic is between τ_6 and τ_7. The τ_6-τ_7 pair is replicated

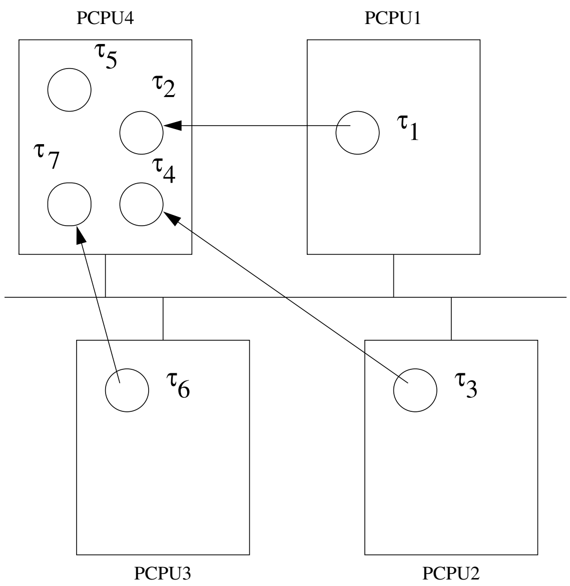


Figure 1: Simulated Task Set

to generate more background traffic. The machines are labeled PCPU1-4; PCPU is an abbreviation for preemptable CPU. Note that the contention occurs on PCPU4.

In the simulation, we vary several parameters in order to determine which protocol processing techniques work best under various ratios of protocol processing time to context switch time. The parameters for the simulation are

- the technique to use for protocol processing,
- the number of background threads (i.e. the size of the spike),
- the context switch time, and
- the protocol processing time.

The techniques that we simulate are T1F, T1P, T1PI, T2P and T4P. The first three are the easiest to implement, and the last two are meant to determine how two levels of preemptability (two threads vs. four threads) compare. The context switch time is the time it takes to move the information for a single task onto or off of the processor. To completely switch from one task to another requires two context switch times in our model, one to save the state of the previous task and one to put the information for the next task onto the processor. The protocol processing time is the time for the message to get from the network up through the protocols to the user or the time for the message to get from the user through the protocols to the network (we assume these are the same for the purposes of this simulation). We have done some measurements using the ARTS Kernel which indicate that this is a reasonable assumption [4].

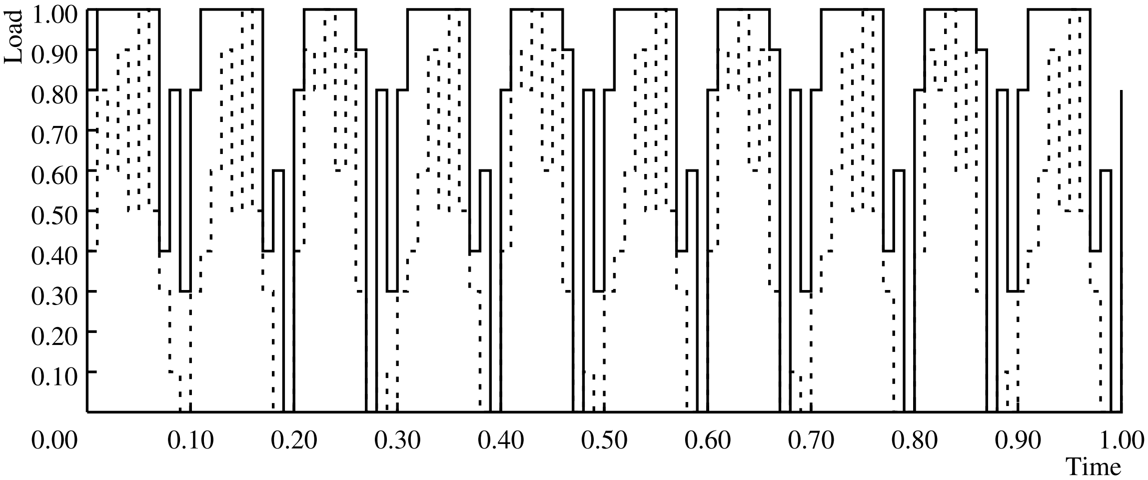
4.2 Simulation Output

The simulation output consists of several measurements. For each of the four tasks described above, we measure the time it takes for the task to finish the job for the current period. This usually consists of sending a message and getting a reply. The actual measurements are taken from τ_1, τ_3, τ_5 and τ_6. All but τ_5 are message generators and for τ_5, we measure the time it takes to finish the computation during each period. We then compute the mean response time, standard deviation, minimum, maximum, and number of missed deadlines for a 10-second interval of the simulation. Since the tasks are periodic, the behavior is deterministic and so a 10-second interval is enough to ascertain the behavior of the system in this case. These statistics are computed separately for each type of task so that we can get an idea of the performance to expect from high priority network activity, high priority local activity, medium priority network activity, and low priority network activity.

We take two other measurements from the simulation. The first is the utilization which we get by computing the length of time the processor is used during some interval and dividing by the duration of the interval. We measure the priority inversion factor in the same way by computing the length of time where the highest priority job is not the one on the processor during some interval and then dividing by length of time that the processor was in use during the same interval. The priority inverted utilization is then the total utilization multiplied by the the priority inversion factor.

4.3 The "Ideal System"

We ran one simulation with a context switch time of 0 and a protocol processing time of 2 ms. We denote the simulation run with the ratio of protocol processing time to context switch time measured in microseconds, 2000-0 in this case. Using these parameters, we can get a feel for how the techniques behave in an "ideal system." In such a system, any priority inversion measured in the system is due to the actual technique; the system does not impose any unavoidable priority inversion as is the case when the context switch time is non-zero (we assume that the processor is not preemptable during the context switch). In the case of the 2000-0 system, the high priority audio task achieved the lowest average response time and lowest variance from the T4P technique although T1P, T1PI and T2P also performed well. T1F resulted in larger and larger variance for this class of activity as the spike size increased. For the local high priority task, T1F and T1P resulted in large and highly



The utilization for the single-threaded FIFO queueing technique. The protocol processing time was 2000 μs and the context switch time was 0. The horizontal axis is time measured in seconds and the vertical axis is the load factor. The solid line indicates the utilization which was calculated for 10 ms intervals, and the broken line describes the priority inverted utilization which is calculated in the manner described in the text.

Figure 2: Utilization (T1F, 2000-0)

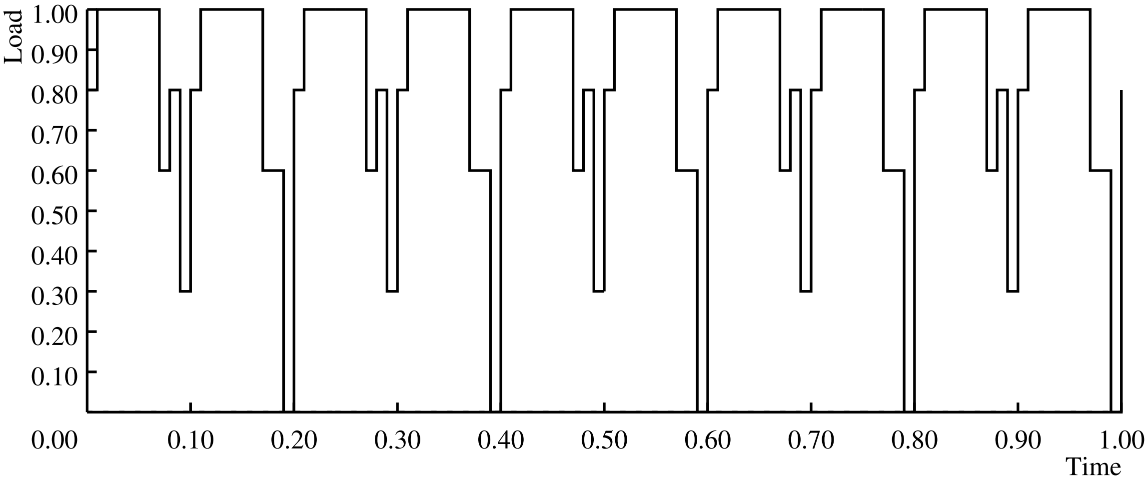
variable response times whereas the other techniques resulted in lower means and low, constant variance. For the medium priority network activity, T1F performed badly, but the others performed reasonably well. And the low priority activity suffers under the prioritized techniques, as expected.

The measurement of utilization and the priority inversion factor is interesting for the 2000-0 system since we can see that the T1F technique results in much more priority inversion than the T4P technique which is optimal in this case and results in a priority inversion factor of zero. This is because with four threads in the protocol processing software and perfect preemptability, it is not possible for the running thread to have a lower priority than the highest priority waiting thread. Figure 2 shows the utilization and priority inverted utilization for a one-second period of the simulation for the PCPU4 processor using the T1F protocol processing technique. The figure shows the case with eight background tasks. The total utilization for PCPU4 is given by the solid line, and the portion of the total utilization under which priority inversion occurs is given by the broken line. Note that the priority inverted utilization is bounded by the total utilization. In the case, the priority inversion factor is 0.609 which means that almost 61% of the time that the processor is working (which is approximately 78% of the time), it is working on the wrong job (i.e. it is not working on the highest priority ready job). That is to say that the processor is working on the wrong job about 48% of the time; we call this 48% the priority inverted utilization. Figure 3 shows the total utilization and priority inverted utilization for the T4P technique. In this case, the priority inverted utilization is zero, and so the broken line does not appear in the figure.

4.4 Trends in Simulation Results

We simulated systems with a context switch time of 100 μs and with ratios varying from 2000-100 down to 100-100. We also simulated systems with a context switch time of 50 μs and with ratios from 2000-50 to 50-50. We found that for ratios of 100-100 and 50-50, the performance is the same over all of the tested techniques. This indicates that when the cost of preemption is roughly equal to the cost of the protocol processing, it does not make sense to preempt. However, when the ratio is 500-100 or 1000-100 or more, then the time-constrained activities suffer high latency and high variance under the T1F and T1P techniques as the spike size increases. These activities also suffer from missed deadlines in the case of the T1F, T1P, and T1PI techniques whereas deadlines are always met in our simulations for the T2P and T4P techniques. Also, we find that under T1F and T1P, the priority inversion factor increases rapidly as the spike size increases compared with the other three techniques.

These results indicate that the T1F and T1P techniques are not suited to this type of workload, while the T1PI, T2P and T4P techniques all perform much better. The T2P and



The utilization for a multi-threaded prioritized queueing technique. The protocol processing time was 2000 μs and the context switch time was 0. The horizontal axis is time measured in seconds and the vertical axis is the load factor. The solid line indicates the utilization which was calculated for 10 ms intervals, but the graph indicates (by the absence of the broken line) that there is no priority inverted utilization in this case.

Figure 3: Utilization (T4P, 2000-0)

T4P techniques are especially suited to ensuring that the time-constrained activities do not miss their deadlines.

We show the results of the simulation for the case where the protocol processing time is 2000 μs and the context switch time is 100 μs. Figure 4 shows the mean, standard deviation and maximum for the response time for the 16-bit audio task. The figure also indicates the number of missed deadlines for this task. Each of the curves corresponds to one of the protocol processing techniques as indicated. The horizontal axis indicates the number of background tasks for the simulation, and the vertical axis gives the measurement in μs or missed deadlines as appropriate. With these graphs, we can get an idea of how the techniques perform relative to each other as the spike size increases.

For the 16-bit audio task (τ_1) in Figure 4, the mean response time is close for all techniques except for T1F which begins to climb slightly. The response time variance for the techniques exhibit similar behavior except the T1F case which again increases. So far, the techniques seem to have been performing equally well, but when we examine the missed deadlines for the various techniques, we see that the T1F, T1P and T1PI techniques are causing deadlines to be missed. The (preemptable) T2P and T4P techniques, on the other hand, are not suffering from missed deadlines for this class of activity. Note that because of our policy of ignoring the computation from a missed deadline, the fact that deadlines are missed in a particular technique is not reflected in the mean and standard deviation numbers for that technique.

In Figure 5 we have the results for the local high priority activity (τ_5). These results clearly indicate that the performance of the T1F and T1P techniques degrades more and more as the number of background activities (or size of the spike) increases. These results indicate the major weakness of the prioritized but non-preemptable techniques.

Figure 6 shows the results from the 8-bit audio task. Here we see behavior similar to that of the 16-bit audio task with the performance of the T1F technique degrading as the spike size increases. But for the 8-bit audio task, no deadlines are missed indicating that the less stringent timing requirements for the 8-bit audio can be honored in this case.

Finally, we consider the effects of these techniques on the background traffic. The T1F technique gives the best service to this activity (at the expense of the time-critical activity). For the techniques which honor the priority assignment, the service for the background traffic is worse. The techniques which give the best service to the high priority activities cause deadlines to be missed in the background activity. For convenience, we model the background traffic as periodic and so we are able to detect the condition where the period of the background traffic is exceeded by the delay. In actual systems, the background traffic will not necessarily be periodic and may have no time constraints. So the fact that response time degrades enough to miss deadlines in this simulation will be reflected only in the

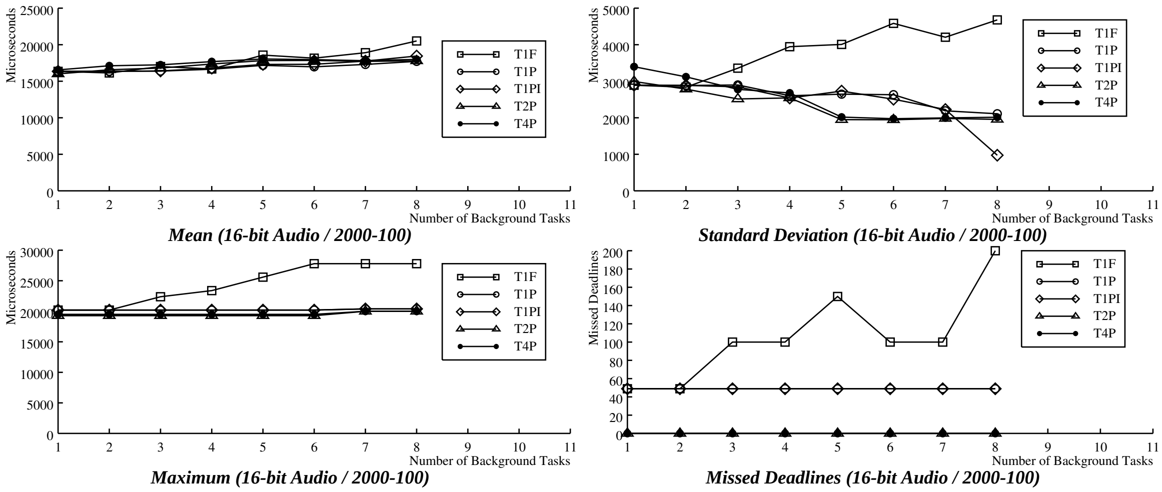


Figure 4: 16-bit Audio Task Performance (2000-100)

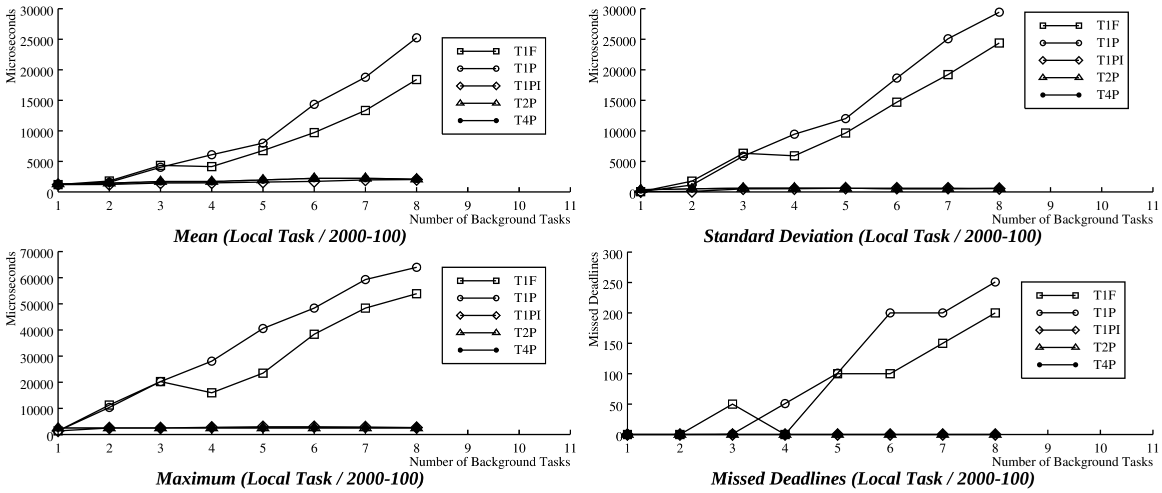


Figure 5: Local Task Performance (2000-100)

response time numbers for the actual system.

5 Discussion

We begin this section with a discussion of the relationship of the priority inversion factor to the theoretical idea of schedulable bound. In formulating our simulation, we made several assumptions about what a representative task set might look like, the characteristics for the background traffic, and the medium-level network model. In Section 4, we indicated how we chose the form of the task set and why we think it is representative of continuous media systems, so here we shall concentrate on the rationale for our model of the background traffic and the network. We then explore the importance of the ratio between context switch time and protocol processing time in choosing a protocol processing technique.

5.1 Priority Inversion and the Schedulable Bound

In the real-time scheduling literature, there is a notion of schedulable bound which is the capacity-based idea of the maximum utilization at which all deadlines are met for a given task set. For example, suppose we have a task set which is composed of *m* preemptable, independent, periodic tasks, and a worst case execution time is associated with each task. We take the deadline of each task to be the beginning of the next period. The rate monotonic priority assignment [11], where higher priorities are assigned to higher frequency tasks, guarantees that all tasks will meet

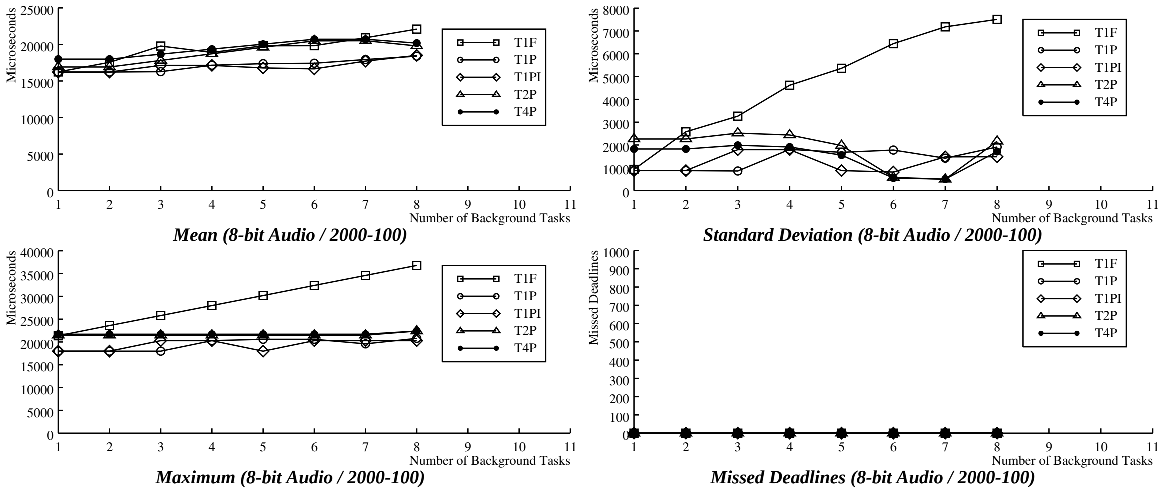


Figure 6: 8-bit Audio Task Performance (2000-100)

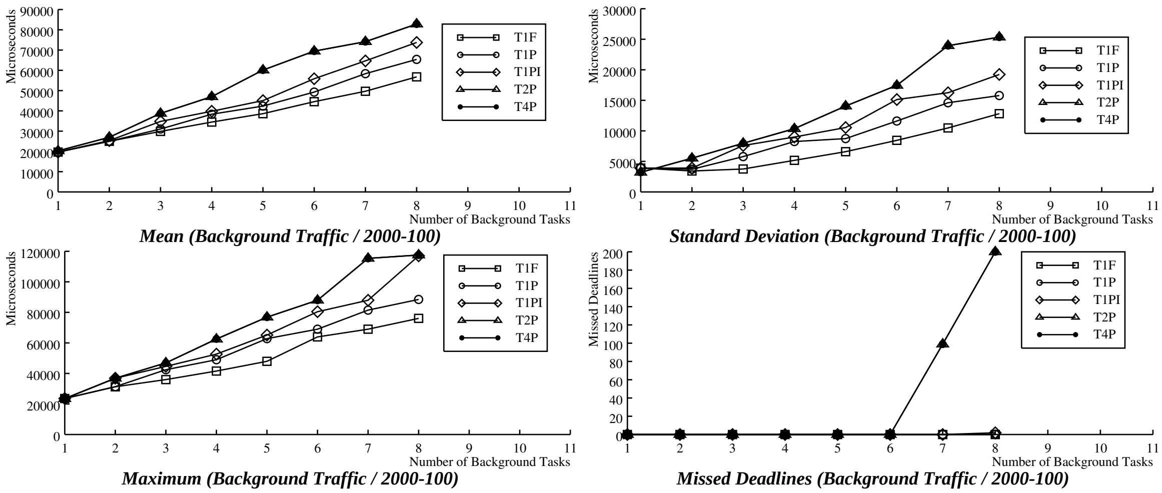


Figure 7: Background Traffic Performance (2000-100)

their deadlines if the total utilization of the task set is less than $m(2^{1/m} - 1)$. This is the least upper bound or the schedulable bound for this task set. If the total utilization exceeds this bound, the task set is no longer guaranteed to meet all deadlines, although for some special cases, the deadlines will still be met.

Now if the tasks are allowed to synchronize with each other, the possibility of priority inversion is introduced. Consequently, the schedulable bound is reduced [15], and the longer the potential blocking time, the more the schedulable bound is reduced. So we see that priority inversion seems to be inversely proportional to the schedulable bound, and bounding the priority inversion in a task set will yield a better schedulable bound, at least for this simple case.

From this notion of schedulable bound and its relationship with priority inversion in simple task sets, we conjecture that the same effects apply for more general task sets. It is more difficult to compute the schedulable bound for these complex task sets, but it seems that the idea of reducing priority inversion to increase performance in terms of the schedulable bound is still valid. Therefore, a measure of the priority inversion of a system would give some indication (at least in a relative sense) about performance of the system in terms of schedulability, and our priority inverted utilization and priority inversion factor provide a way to quantify this concept.

5.2 Formulation of the Simulation Model

It is clear from our experience that the current generation of time-sharing workstations and networks such as Ethernet perform well enough to support some very limited animation and audio processing applications. The applications work well most of the time but frequent "glitches" occur which can be merely annoying or can be devastating to the desired effect of the system. One of our goals is to find the source of these glitches and to find methods to structure the system which avoid or reduce the effects of these glitches. Glitches occur when the system experiences more demand for a resource (such as processor time) than there is supply, and in order to study the behavior of various algorithms or scheduling policies under these high demand situations, we create "spikes" of high demand periodically. Thus the performance results obtained from these tests cannot predict actual mean response time in a typical system, but we contend that the results are indicative of the expected performance of the system under strenuous conditions.

In our simulation, we model the network as a non-prioritized, non-preemptable resource. The time to transmit a single message was taken to be about 1 ms for the purposes of this study. We chose this model because it reflects the capability of the current technology for LANs (e.g. Ethernet). The central goal of this study is to determine how the protocol processing technique affects the performance of distributed applications without attacking issues related to the media access level of the network. One alternative choice would be to model the network as an infinite capacity resource with instantaneous message transmission. This would tend to remove any effects of the network scheduling from the results, but then it would be more difficult to verify the results on our testbed (which we intend to do). Therefore, we choose to model the network technology which is most prevalent and readily available.

5.3 Context Switch Time vs. Protocol Processing Time

The ratio between the context switch time and protocol processing time is of critical importance in motivating the choice of protocol processing technique. Intuitively, we reduce priority inversion by increasing the preemptability of the network software, but if the duration of protocol processing is small compared with context switch time, it makes sense to finish the protocol processing without preempting. This study is intended to determine the performance of a distributed application for different values of this ratio.

The results reported in Section 4 about the behavior of the system under different ratios give some indication of what we can expect if we decrease the protocol processing time by software optimization or if we decrease the context switch time by implementing a context switch instruction in hardware.

Many efforts in the communication field focus on the hardware implementation of protocols [16] or the use of communications co-processors to speed communication and free the main CPU from communications-related tasks [6]. The results of this study are applicable to these configurations as well. The simulated task set includes a local high-priority computation to account for the interference between the protocol processing workload and the normal local workload, and many of the results reflect the priority inversion which arises from this interference. However, communication co-processors will still have to service messages with different priority levels, so it still makes sense to consider how the ratio between context switch time and protocol processing time would affect overall performance. Thus, some of the lessons learned from this study can be applied to the more advanced architecture.

In the case of the communication co-processor, the relative speed of the communication processor and the host processor is important as well. If the communication processor is much faster than the host processor, priority inversion in the protocol processing will not have as large an effect as when the communication processor is similar to the host processor in speed.

6 Related Work

We have implemented several of the protocol processing techniques in the ARTS Kernel [18], a real-time operating systems testbed developed at Carnegie Mellon. In [9], we report that the multi-threaded technique has the most predictable performance and this technique imposes only an additional 10% overhead compared with the software interrupt technique which is not so predictable.

The idea of schedulable bound has been used in formulating a benchmark suite for real-time systems [12] where we define benchmark task sets, each with a single variable parameter which is related to the schedulable bound. We increase the value of this parameter to increase the utilization for the task set; the idea is to find the largest parameter value (i.e. the largest utilization) at which the system can execute the given task set without missing deadlines. In this way, we experimentally determine some relative measure which is related to the schedulable bound for each task set.

In [7], several media access level protocols are simulated and the performance results given for voice, interactive data, bulk data and facsimile data packets utilizing a single network. The priority granularity problem surfaces for the case where only three priorities are allowed for the four classes of packet. A case of priority inversion is described where voice packets miss their deadlines when a long facsimile transmission occurs on the network. The technique developed to alleviate this problem is to artificially raise the contention priority of the voice packets that have been waiting the longest after the facsimile transmission completes. These results focus on the media access level whereas our results concentrate on the upper level protocols where these effects may be more pronounced due to longer protocol processing times.

Other protocol implementations, such as the *x*-kernel [8], have used the multi-threaded approach for organizing message processing. The *x*-kernel threads are also prioritized, but preemption is not allowed while the thread is executing the protocol. This non-preemptability is imposed in order to take advantage of caching and other optimizations, and introducing preemptability would not only make the coding of the protocols more complex, but would also negate some of the optimizations. One objective of our study was to determine the value in terms of predictability of prioritized, non-preemptable processing compared with prioritized, preemptable processing. The study provides evidence that, even under the T1P and T1PI techniques which are prioritized but non-preemptable, background spikes can cause the time-constrained activities to miss deadlines whereas the T2P and T4P multi-threaded techniques effective guard against these missed deadlines for the high priority activity.

The Nectar system [6] employs a communication processor in addition to the host CPU. The idea is to reduce the demand on the host processor for protocol processing. The Nectar CAB (communication accelerator board) supports both protocol processing and application level processing. Currently, the implementation does most of the protocol processing in the interrupt handlers, but a new implementation with the protocol processing done in high priority threads is under development. Our results show that a prioritized thread approach would be better than a high priority thread approach especially since application threads may be resident on the CAB.

A new protocol, SRP [2], for continuous media communication has recently been proposed. SRP is a resource reservation protocol intended to guarantee that the performance of time-constrained message traffic is maintained within the bounds of the requested parameters. The DASH resource model [3] is used to determine the resource requirements to be reserved. The issue of choosing a protocol processing technique is raised in [1]. Our study complements this work in that we show how the intentions of the reservation scheme can be enforced without problems of priority inversion interference, and furthermore, we consider the advantages and disadvantages for a wide range of protocol processing structures.

Several protocols include priority fields in the message format which is essential for the prioritized processing we describe in our study. VMTP [5] and XTP [16] are examples of prioritized protocols. In fact, XTP supports 2^{32} priorities. A VLSI implementation of XTP promises much better basic performance than current software-based protocol implementations, and in our study, we have shown that prioritized protocol processing techniques are equal in terms of performance if the protocol processing time is small compared with the context switch time.

7 Conclusion

Future operating systems will have to be able to effectively handle time-constrained audio and video applications as well as traditional non-time-constrained programs. The protocol processing software is an area where prioritization and preemptability are critically important to the performance of continuous media applications. We have studied the efficacy of various protocol processing techniques and have found that a prioritized, preemptable protocol processing engine is essential for the predictability of these types of systems. These results shed new light on the engineering tradeoff between highly optimized, non-prioritized, non-preemptable protocol processing vs. highly optimized, prioritized, preemptable protocol processing. In the case of the former, high performance is obtained at the expense of predictability while the latter sacrifices some raw performance in favor of predictability for time-constrained activities. The ratio between the protocol processing time and the context switch time is instrumental in determining which technique is best for a particular system.

Our results are based on the observation that the mean is not the only important measure of system performance. Variance is also critically important for evaluating the predictability of a continuous media system. We also introduced an important metric which we call the priority inversion factor which is related to resource utilization. The priority inversion factor is a measure of the fraction of total utilization in which priority inversion occurs.

Acknowledgements

We would like to thank Stefan Savage for his comments on earlier drafts of this paper and all the members of the ART Project at CMU for providing such a stimulating working environment.

References

[1] D. P. Anderson, L. Delgrossi, and R. G. Herrtwich. Structure and Scheduling in Real-Time Protocol Implementations. Technical Report TR-90-021, International Computer Science Institute, June 1990.

[2] D. P. Anderson, R. G. Herrtwich, and C. Schaefer. SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet. Technical Report TR-90-006, International Computer Science Institute, February 1990.

[3] D. P. Anderson, S. Tzou, R. Wahbe, R. Govindan, and M. Andrews. Support for Continuous Media in the DASH System. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 54–61, May 1990.

[4] N. Ashida and Tokuda H. Persistent File/Object Management for Hard Real-Time Operating Systems. Unpublished manuscript, 1990.

[5] D. R. Cheriton. VMTP: Versatile Message Transaction Protocol. Technical report, Computer Science Department, Stanford University, January 1987.

[6] E. C. Cooper, P. A. Steenkiste, R. D. Sansom, and B. D. Zill. Protocol Implementation on the Nectar Communication Processor. In *SIGCOMM '90 Symposium on Communications Architectures & Protocols*, pages 135–144, September 1990. Also available as CMU Technical Report CMU-CS-90-153.

[7] S. S. Gaitonde, D. W. Jacobson, and A. V. Pohm. Bounding Delay on a Multifarious Token Ring Network. *CACM*, 33(1), January 1990.

[8] N. C. Hutchinson and L. L. Peterson. The *x*-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, January 1991.

[9] Y. Ishikawa, H. Tokuda, and C. W. Mercer. Priority Inversion in Network Protocol Module. *Proceedings of 1989 National Conference of the Japan Society for Software Science and Technology*, October 1989.

[10] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, 1989.

[11] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *JACM*, 20(1):46–61, 1973.

[12] C. W. Mercer, Y. Ishikawa, and H. Tokuda. Distributed Hartstone: A Distributed Real-time Benchmark Suite. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, May 1990. Also available as Techical Report CMU-CS-90-110.

[13] C. W. Mercer and H. Tokuda. The ARTS Real-Time Object Model. In *Proceedings of 11th IEEE Real-Time Systems Symposium*, December 1990.

[14] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for Some Practical Problems in Prioritized Scheduling. In *IEEE Real-Time Systems Symposium*, December 1986.

[15] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9), September 1990.

[16] R. Simoncic, A. C. Weaver, and M. A. Colvin. Experience with the Xpress Transfer Protocol. In *15th Conference on Local Computer Networks*, pages 123–31, September 1990.

[17] R. Steinmetz, R. Heite, J. Rückert, and B. Schöner. Compound Multimedia Objects – Integration into Network and Operating System. In *Proceedings of First International Workshop on Network Operating System Support for Digital Audio and Video*, November 1990.

[18] H. Tokuda and C. W. Mercer. Arts: A distributed real-time kernel. *ACM Operating Systems Review*, 23(3), July 1989.

[19] H. Tokuda, C. W. Mercer, and S. E. Breach. The Impact of Priority Inversion on Continuous Media Applications. In *Proceedings of the International Workshop on Network Operating System Support for Digital Audio and Video*, November 1990. Available in International Computer Science Institute Techical Report TR-90-062.

[20] H. Tokuda, C. W. Mercer, Y. Ishikawa, and T. E. Marchok. Priority Inversions in Real-Time Communication. In *Proceedings of 10th IEEE Real-Time Systems Symposium*, December 1989.