Workstation Support for Real-time Multimedia Communication*

Olof Hagsand and Peter Sjödin
Swedish Institute of Computer Science

Abstract

We show how multimedia applications with real-time requirements can be supported in a distributed system. A UNIX system has been modified to give soft real-time support. The modifications include deadline-based scheduling, preemption points and prioritized interrupt processing. In addition, a system call interface for real-time application programming has been designed. We justify the modifications by experiments with a simple distributed multimedia delivery system. The experiments are made on an ATM network, where resources are reserved by means of the ST-2 internetworking protocol.

1 Introduction

Modern workstations support multimedia interaction. They are (or can be) equipped with loudspeakers, microphones, video cameras and high-resolution displays. Such devices are different from "traditional" computer I/O-devices since they are *continuous*, which require special treatment by the workstations. Consider as an example of a distributed multimedia application the "netphone", an application for audio communication between workstations (see Figure 1). Each workstation has two processes, *mike* and *speaker*. The mike process collects sound samples from the microphone, puts them in packets and sends them on the network. The speaker process receives sample packets from the network, unpacks the sound samples and feeds them to the loudspeaker.

The netphone is an application for human communication and therefore has stringent timing requirements. If the delay from microphone to speaker is too long, the round-trip time between the two users will make the netphone annoying to use. If sound samples are not fed to the loudspeaker at regular intervals, the sound will be distorted and the netphone will be difficult, or even impossible, to use. Thus, the netphone has requirements on delay as well as on delay variation. It is hard to define an upper bound on delay for voice communication, 100–250 milliseconds has been suggested [HSS90, And93]. Other applications, such as distributed musical systems, may require delays down to a few milliseconds.

Traditional time-sharing operating systems fail to provide adequate support for real-time multimedia communication. In such systems, the more processes there are, the more seldom each process executes. For many applications, this is acceptable—it just takes longer time for them to fulfill their tasks on machines with heavy load. It is not acceptable for the netphone, however, since the mike and speaker processes need to meet their deadlines even when the load is high.

Our experiences with netphone and similar applications are that when the load increases, they are executed in a way that renders them practically useless. It should be pointed out that the problem with general-purpose workstations is not the hardware performance—on machines with light load, multimedia applications run without problems. Similar experiences have been made, for instance, with the ACME multimedia server on a Sun workstation [GA91].

Real-time operating systems are designed for applications with strict timing requirements, so we could use such a system for the netphone. Other researchers have demonstrated that it is possible to use real-time

*to appear in USENIX Winter 1994 Technical Conference operating systems, for instance the YARTOS operating system [JSS92], for multimedia communication. Real-time operating systems are often oriented towards command and control systems. These systems have so called *hard* deadlines, meaning that the cost of missing a deadline is very high. Distributed multimedia have more relaxed timing requirements. For instance, from experiments with two-way video communication, it is reported that 4 video frames out of 1000 can be lost without significant loss in of quality [JSS92]. We think that multimedia capabilities should be integrated in existing workstations, and therefore prefer to study whether current workstations can be modified to better support multimedia.

In this paper, we determine how an existing operating system can be modified to allow multimedia applications to execute undisturbed by concurrent activities. Specifically, we wanted to show how netphone could be made useful in a networking environment. The experiments were made on Sun SPARCstations running SunOS 4.1.1, which, in all aspects relevant for this paper, is equivalent to 4.3BSD UNIX [LMKQ89]. The Sun SPARCstations are connected by an ATM network.

In Section 2, we describe and motivate support for real-time multimedia applications in end-systems and networks. Section 3 describes the necessary modifications of the SunOS kernel. Then, in Section 4, we report and discuss measurements of a simulated multimedia implementation, and Section 5 concludes the paper.

2 Real-time for distributed multimedia systems

A common model of distributed multimedia applications is to regard them as sets of coordinated continuous streams between input and output devices. Although a convenient conceptual model, it is not an adequate description of a multimedia system from a computer system point of view. The main problem is the continuous streams—general-purpose computers are highly asynchronous, and handle data in chunks.

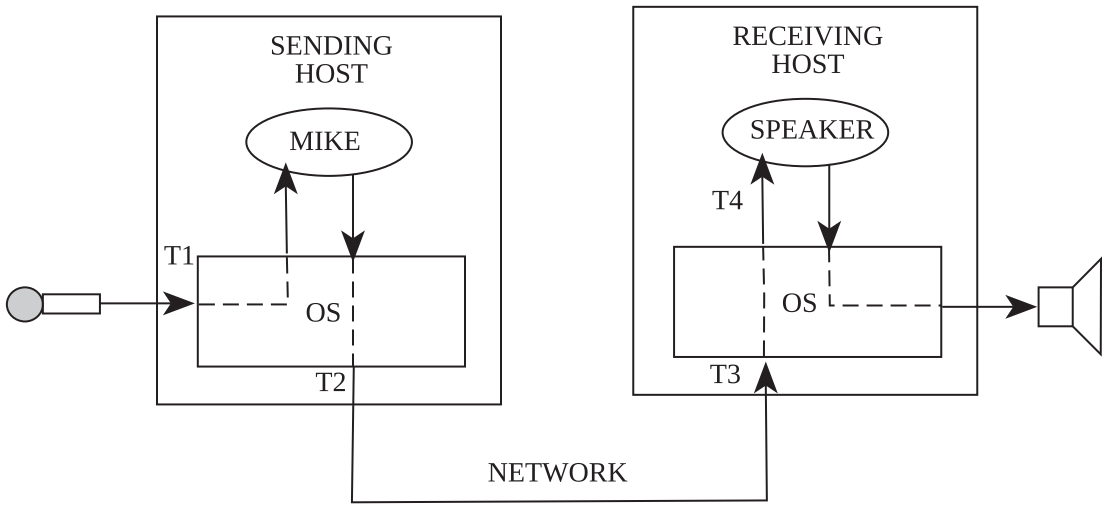

Figure 1: Communication between one microphone and one speaker in the netphone application.

In this paper, we consider simplified multimedia systems (exemplified by netphone) consisting of two hosts: a source and a destination. When data is available on an input device at the source, a process collects the data, processes it, and sends it on the network. At the destination host, the data arrives to a process that feeds it to an output device.

The multimedia application tries to provide a constant delay between the input and output devices (the input-output delay). The input-output delay is determined by the processing time in the sender and destination processes, the scheduling delay in the source and destination hosts (the time a runnable process is blocked from execution), and the network delay. The application is based on an assumed worst-case input-output delay, and the application delays artificially all data items that can be fed to the output device with a delay shorter than the worst-case. In this way, the multimedia application guarantees a constant input-output delay equal to the assumed worst-case delay, regardless of jitter in the network, for example.

We distinguish between two types of delay: application-specific delay and system delay. The application-specific delay can be caused by copying of data to and from multimedia devices, processing in the sender and destination processes, and so on. The system delay consists of network delay and scheduling delay. Our aim is to reduce the worst-case input-output delay for multimedia applications in general, and therefore we focus on the system delay and do not consider the application-specific delay.

The above is a description of unicast (single destination) applications, but we think it applies to multicast (multiple destinations) applications as well. We assume that the distribution of multicast messages is taken care of entirely by the network, so the hosts and processes in a multicast application can operate as if it was a unicast application. Under this assumption, we do not need to distinguish between unicast and multicast applications.

We use for our experiments a simplified version of the netphone that has a minimal amount of application-specific delay (it sends "dummy" data instead of sound samples from the microphone). Our goal is to limit the system delay to 10 ms. An input-output delay of 100 ms seems to be acceptable for real telephony applications, which leaves 90 ms for application-specific delay. This appears to be realistic, even for CD-quality sound. Video communication has roughly the same delay requirements, so 10 ms scheduling and network delay would be sufficient also for video. However, in our experience, video communication has high application-specific delay, since it requires higher throughput and more processing. In addition, special hardware support is needed to achieve medium-quality video communication.

2.1 Real-Time Support in the Scheduler

We are interested in workstations in a networking environment with many sources of disturbances. Operating systems must process interrupts spawned by other activities on the machine, other processes on the machine execute, networks introduce latencies, etc.

We cannot make any assumptions about when disturbances occur, and we cannot predict exactly when communication events take place in the multimedia system. This has lead us to implement support for event-driven real-time processes. Such processes are triggered by external events, rather than an internal clock. In the netphone, the external events are arrivals of network packets on the destination side, and sound samples on the source side. When an event occurs, the process becomes runnable and needs to perform its task before a *deadline* expires. A runnable real-time process that has not yet met its deadline is *critical*.

2.2 Real-time Support in Networks

In order to maintain an upper bound on network delay, there must be some support for real-time in the network. Ethernet, for example, does not provide this. A packet on an Ethernet can be delayed for an unpredictable amount of time due to other network traffic. We have therefore chosen to experiment with an ATM network (from Fore systems Inc.), which provides practically constant network delay.

ST-2 [Top90] is an experimental internetworking protocol for multimedia applications. It is a connection oriented protocol (point-to-multipoint) where *streams* are constructed from one originator to a set of recipients. A stream is constructed through resource reservations (formulated as flow specifications) that are passed from the sender to the network. ST-2 allows the sender to reserve resources on the ATM network, by passing flow specifications from the application to the resource reserving facilities in the ATM network via the ATM signalling protocol. We have used SICS' implementation of ST-2 [PP91] and interface to ATM [HP93].

3 Modifications of the Operating System

The main effort of our modifications was to implement real-time support in the operating system and design a programming interface for real-time processes. Much of this work was influenced by RT-MACH [TNR90].

3.1 Scheduling

Our intentions were to modify an existing system to incorporate deadline-based scheduling of real-time processes. We chose to implement the earliest deadline first (EDF) scheduling algorithm [LL73]. EDF is well-suited for event-driven aperiodic processes since it does not require real-time processes to be periodic. Essentially, EDF schedules processes by giving the highest priority to the process with the closest deadline.

A critical process runs at such a high priority that it cannot be interrupted by non-critical processes. It runs until it voluntarily yields control of the processor, the deadline expires, or a new critical process appears with a deadline that is closer in time. Our goal is to minimize startup time and the effects of disturbances during execution.

Our method is to make only minor modifications to the existing scheduling code. When a critical process is scheduled for execution by the deadline scheduler, it is put at the head of the highest priority run queue before a context switch is invoked. This guarantees that the critical process will run next. The actual context switch code can therefore be left unmodified. When a critical process yields control of the CPU, the deadline scheduler is invoked to find the next critical process, if any.

3.2 Application Programming Interface

We designed a programming interface for real-time processes based on three new system calls: `rt_select` and `rt_periodic` to launch aperiodic and periodic real-time processes, respectively, and `rt_deadline` to signal that a critical process has met its deadline (and thus turned non-critical).

Our event-driven interface is the `rt_select` system call, which is based on the SunOS 4.1 `select` system call. A process calls `rt_select` to register file descriptors that should trigger critical periods. The process then calls `select`. If an event occurs on one of the file descriptors registered in the `rt_select` system call, the process turns critical. The process stays critical until it either calls `select` again, calls `rt_deadline` to signal that the deadline is met, or misses the deadline. The syntax of `rt_select` is as follows:

```
int rt_select (width, readfds, writefds, exceptfds, deadline, options)
int width;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *deadline;
int options;
```

where `readfds`, `writefds` and `exceptfds` indicate the file descriptors that should trigger critical periods. `Deadline` specifies the maximum time the process may be critical before a deadline expires. The `options` field specifies how the process should be notified of a start of a critical period and deadlines, etc.

The interface also contains a periodic real-time interface, where a periodic process is initiated with a call to `rt_periodic`. The syntax of `rt_periodic` is as follows:

```
int rt_periodic (value, period, deadline, options)
struct timeval *value;
struct timeval *period;
struct timeval *deadline;
int options;
```

where `value` specifies when the first period begins, `period` specifies the time between each start of a critical period and `deadline` specifies the maximum time the critical task may execute after a critical period starts.

We have used the UNIX signal facility to indicate beginning and end of critical periods. A process can then register signal handlers to be invoked at the start of a critical period and a missed deadline, respectively. The process indicates that it has finished its critical period either by calling `rt_deadline`, or by the normal return of the signal handler. Missing a deadline in a soft real-time system is not considered as catastrophic. However, we have chosen to lower the priority, so that a missed deadline will not affect the execution of other critical processes.

3.3 Preemption Points

We found the above changes to be insufficient for supporting multimedia on machines with concurrent activities—it can still take long time before a critical process starts running. This is because context switches cannot occur at any time in SunOS 4.1. In particular, the SunOS 4.1 kernel is not preemptive, which means a process operating in kernel mode (executing a system call) cannot be preempted at all. Some system calls can take long time to execute, and thus block critical processes for long periods of time. We tried to identify such system calls and insert *preemption points* into them. At a preemption point, a (non-critical) process checks whether there is a critical process. If there is, the process yields the processor and thereby preempts itself.

System calls that take long time are typically calls that iterate through a set of objects, and perform some operation on each object. For example, the `exit` system call loops through the memory pages of a process and releases them, one by one. This takes a long time for processes with many pages. Other significant examples are truncation of large files (in the `open` system call), flushing the file system cache (`sync`), copying process memory (`fork`), and loading executable files (`execve`).

Inserting preemption points is a simple method of limiting the blocking time of critical processes. A drawback is that it is often necessary to put preemption points in the bodies of loops. Such preemption points will execute often, and may slow down the system. This illustrates a trade-off between real-time guarantees and execution time; by slowing down the system, we limit the time a critical process can be blocked.

A more general solution is to make the kernel "fully" preemptive, as has been done with SunOS 5.0 [KSZ92]. In a fully preemptive kernel, processes can be preempted at any place. Critical regions in the kernel still need to be protected, so fully preemptive kernels need mechanisms for mutual exclusions (e.g. locks and semaphores). This means that whenever a process wishes to enter a critical region, it must first execute some code to ensure exclusive access to the region. Thus, fully preemptive kernels also trade execution time for real-time properties.

3.4 Interrupt Activity

We found interrupt handling to be another source of disturbances for real-time processes. Interrupt handling routines can be thought of as high-priority processes. Once an interrupt occurs, the interrupt handler starts executing and preempts any process that is running.

In SunOS 4.1, interrupt processing may be done at different priority levels. The hardware interrupt handler does the urgent interrupt processing, and then schedules the software interrupt process, "softint", to do interrupt processing at lower priority. For example, a network driver working at a high priority level enqueues incoming packets and re-initializes network interface hardware, but schedules softint to do the protocol processing for the incoming packets.

The softint processing is done at interrupt level, which means that it preempts any process that is running. Much of the softint processing, in our opinion, is less urgent than critical processes, and can be postponed until there are no critical processes. There is, however, softint processing that need to be done of immediately, for instance processing of incoming data for real-time applications (such as incoming audio packets for the netphone).

As an experiment we chose to assign a lower priority to IP packets arriving from the network interface than to ST-2 packets. If there are critical processes, processing of IP packets is postponed. This intermediate solution is based on the assumption that ST-2 packets carry real-time data, whereas IP packets do not. Although this is true for the experiments we have conducted, it is not a sound assumption in general. One would rather prioritize real-time data independently of the protocol that carries it over the network.

The question is if it is possible to discern at interrupt level whether or not incoming data is real-time data? It seems possible to use `rt_select` for this: In the networking case, a file descriptor has an associated protocol control block with an *end-point identifier*. For TCP the destination port could be used as end-point identifier, and for ST-2 the hop identifier.

The idea here is that it is easy to extract the corresponding end-point identifier from a packet's header. For each incoming packet the header is examined, and if it matches a protocol control block that is associated to a "real-time" file descriptor, softint is called immediately, otherwise it is postponed.

4 Experiments

In this section, we describe some experiments made in order to verify that our real-time modifications improve real-time performance.

Two SUN 4/65 (SPARCstation 1+) are connected by a local area ATM 100 Mb/s network, consisting of two 8×8 ATM switches from Fore Systems Inc. The workstations are also connected by an Ethernet. The experimental application is a simplified version of netphone (see Figure 1) where all audio device dependencies are removed. In this version, the mike process sets up an ST-2 connection, generates data and sends packets at a fixed rate. Each packet is timestamped four times (see Figure 1):

T1. When a triggering event (i.e., a timeout) occurs at the sender.

T2. When the packet is sent on the network.

T3. When the packet is received from the network in the receiving machine.

T4. When packet is delivered to the speaker process.

The timestamps are saved and then analyzed off-line, while the data itself is dropped at the destination.

The two workstations are synchronized through NTP [Mil92], but the synchronization is not accurate enough to give an absolute value of the network delay (i.e., the difference between T3 and T2). We therefore assume a fixed network delay of 1 millisecond, which is a slight exaggeration.

In the experiments, we try to use bit-rates typical for multimedia applications. We use two bit-rates, 1.2 Mb/s and 16 Mb/s. 1.2 Mb/s is to mimic audio and compressed medium-quality video [ISO92]. Here, 1500-byte packets are sent at a rate of 100 Hz. 16 Mb/s represents uncompressed video and low quality compressed HDTV. For 16 Mb/s, we have to use a SPARCstation 2 as receiver, an increased rate of 500 Hz, and packet size of 4K bytes. All experiments run over a period of 60 seconds. In the 1.2 Mb/s case, we make a comparative study of transfer over Ethernet.

Our goal is to minimize the effects of disturbances on multimedia communication. It is therefore necessary to apply disturbances to evaluate the real-time modifications. We apply a competing activity on both machines that is both CPU and I/O intensive. This activity performs a compilation, using `make`, of a C program where the sources are stored remotely on an NFS file system. Therefore, there is I/O activity over an Ethernet and a local disc. In addition, all experiments are made under normal working conditions, so there is other traffic on the networks, ordinary background activities on the workstations, and so on.

Table 1 shows the *end-to-end delay* (T4 − T1) for six different experiments. In the table, the minimum, maximum and average delay are given, as well as delay variance. There are four experiment parameters in the table:

RT A "+" indicates that all real-time kernel modifications were turned on at both sender and receiver. A "−" indicates an unmodified SunOS kernel. In the latter case, the mike process is triggered by an interval timer, and speaker blocks in a read loop.

Experiment	Options				Min	Max	Mean	Variance
number	RT	Net	PP	Rate	[10^{-3}s]	[10^{-3}s]	[10^{-3}s]	[10^{-6}s^2]
1	+	ATM	+	1.2	2.5	10.0	3.3	0.3
2	–	Ether	–	1.2	2.7	445.0	16.8	2224.2
3	–	ATM	–	1.2	2.4	188.5	5.3	136.8
4	+	Ether	+	1.2	2.6	39.9	3.3	1.7
5	+	ATM	–	1.2	2.5	103.2	3.3	5.3
6	+	ATM	+	16.0	2.1	26.5	3.1	3.8

Table 1: End-to-end delay (T4 – T1).

Net The network is either ATM or Ethernet.

PP A "+" indicates that the preemption points in the kernel are enabled. This option can only be set if the real-time kernel is used.

Rate The transfer rate is given in Mb/s.

Experiment number 1, over ATM and with kernel real-time modifications, shows that we barely reached our goal to keep input-output delay below 10 milliseconds. Experiment 6, over Ethernet and without kernel real-time support, is the "worst" possible and shows much higher maximum and average delay.

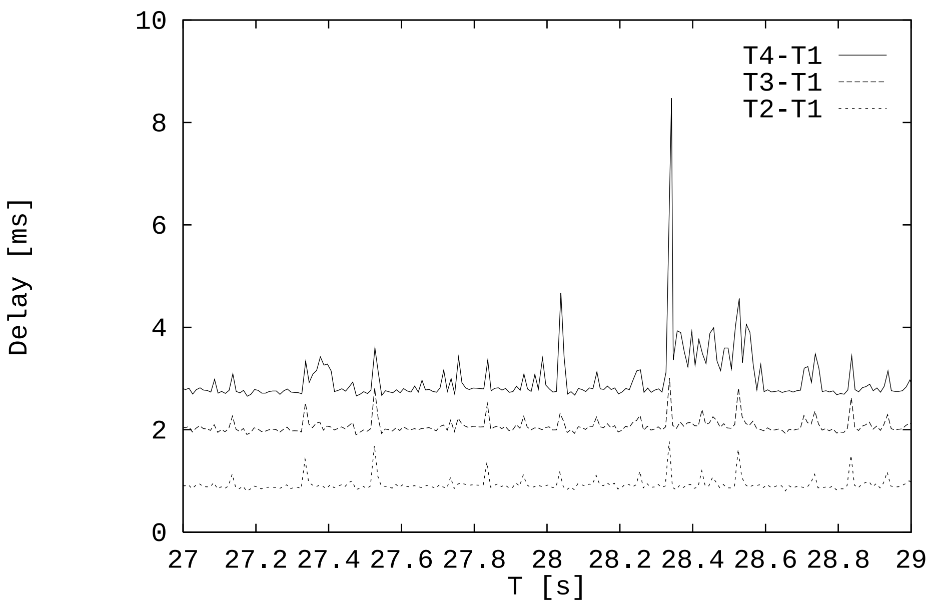

Figure 2: End-to-end delay on ATM with kernel real-time modifications (experiment 1).

Figure 2 and Figure 3 show delay for packets sent during two seconds of experiment 1 and 2, respectively. In these experiments, packets are sent at ten millisecond intervals. The x-axis (in seconds) is the relative time T from the start of the experiment, ranging at most from 0 to 60 seconds. We picked different time intervals from experiment 1 and 2, and therefore the range on the x-axes differ. The y-axis is the difference in time between a timestamp and the first timestamp, T1. That is, for each packet we plot, versus T, the difference in time (the delay) between the time of the triggering event at the sender (T1) and the time when the packet enters the network (T2), leaves the network (T3) and arrives at the speaker process (T4). In both figures, we can see that many packets are delivered with low delay. Some packets are delayed more due to disturbances, represented as spikes in the diagrams. Figure 3 shows that disturbances occur at all places in the system: at the sender, in the network, and at the receiver.

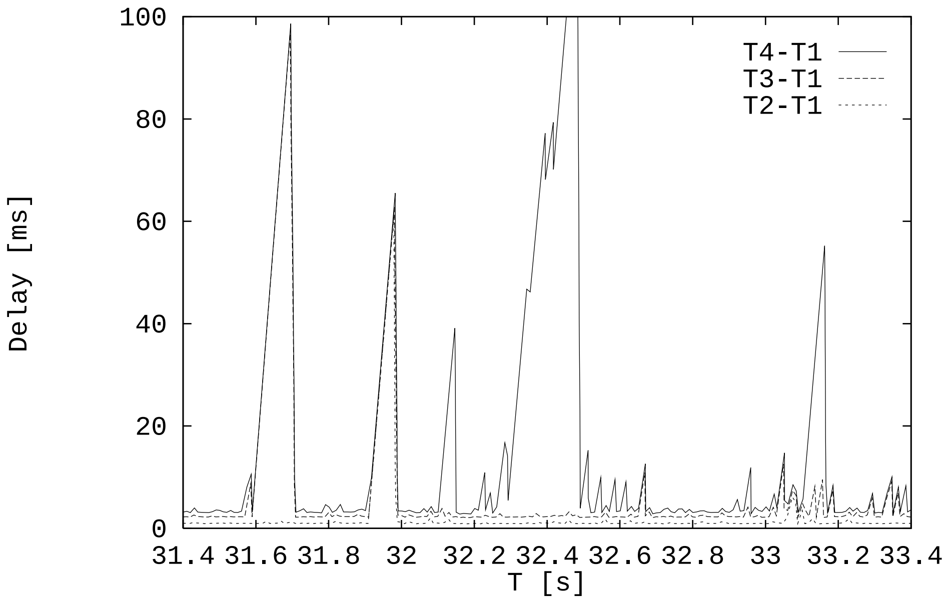

Figure 3: End-to-end delay on Ethernet without kernel real-time modifications (experiment 2).

The third and fourth experiments are included for studying the individual effects of using ATM and our kernel modifications. We see that the maximum delay gets significantly longer both if we use ATM but remove the kernel modifications (experiment 3), and if we keep the kernel modifications but switch to Ethernet (experiment 4). From this we conclude that it is necessary with real-time support in both network and operating system—if only one of them supports real-time, we loose end-to-end real-time properties.

The fifth experiment is made with preemption points turned off. Comparing this with experiment 1, in which preemption points are on, shows that preemption points significantly improve worst-case delay.

In experiment 6, a faster CPU is used as a receiver. Still, the CPU is not fast enough to hinder the buffer buildup which occurred between the socket layer and the speaker process, which we think is the explanation for the increase in maximum delay.

4.1 Prioritized Interrupt Processing

Experiment	Options	Min	Max	Mean	Variance
number	Postpone IP	$[10^{-3}s]$	$[10^{-3}s]$	$[10^{-3}s]$	$[10^{-6}s^2]$
7	+	2.1	7.3	3.0	0.3
8	−	2.1	84.5	5.8	42.1

Table 2: End-to-end delay (T4 − T1) with and without prioritized interrupts.

The disturbances are not enough to show effects on postponing IP input processing, as described in Section 3.4. To show such effects, we generate intensive IP traffic from an external source to the receiving host.

Table 2 shows the end-to-end delay for the two cases. As can be seen from the table, maximum delay increases from 7.3 to 84.5 milliseconds if IP input processing is not postponed. Even though this represents an extreme background load, we have shown that IP input processing can disturb critical processes significantly.

5 Conclusions

We have demonstrated that it is possible to modify a standard UNIX (SunOS 4.1) operating system to provide adequate support for multimedia communication. We have modified the scheduler, added three system calls, made the kernel (slightly) preemptive by using preemption points, and modified interrupt handling. We have verified the effect of the changes by measuring the netphone application on an ATM network, both with the modified and the original kernel. The result is encouraging—with our modifications, we limit end-to-end delay to less than 10 milliseconds, and the netphone runs smoothly even on machines with heavy load.

We conclude from the experiments that in order to get real-time communication from end to end, one must have real-time support in the operating system as well as in the network. When we used an operating system without our kernel real-time modifications, or switched from ATM to Ethernet, the worst-case delay increased several times, giving an end-to-end delay much above the maximal 10 milliseconds.

We have suggested a way of improved interrupt processing in the presence of critical processes. As future work, we plan to implement and analyze this improvement. Furthermore, there are other operating systems available with real-time support, and we would like to study how well they support an application such as the netphone.

References

[And93] D. P. Anderson. Metascheduling for continuous media. *ACM Transactions on Computer Systems*, 11(3):226–252, August 1993.

[GA91] R. Govindan and D. P. Anderson. Scheduling and IPC mechanisms for continuous media. In *Proceedings of ACM Symposium on Operating Systems Principles, ACM Operating Systems Review*, volume 25, pages 68–80, October 1991.

[HP93] O. Hagsand and S. Pink. ATM as a link in an ST-2 internet. In *Fourth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '93)*, Lancaster, November 1993.

[HSS90] D. B. Hehmann, M. G. Salmony, and H. J. Stüttgen. Transport services for multimedia applications on broadband networks. *Computer communications*, 13(4):197–203, May 1990.

[ISO92] ISO/IEC. Information technology — coding of moving pictures and associated audio for digital storage media up to 1,5 Mbit/s, 1992. Draft International Standard ISO/IEC DIS 11172.

[Jac90] V. Jacobson. 4BSD TCP 'header prediction'. *ACM SIGCOMM, Computer Communications Review*, 20(2), April 1990.

[JSS92] K. Jeffay, D. L. Stone, and F. Donelson Smith. Kernel support for live digital audio and video. *Computer communication*, 15(6):388–395, July/August 1992.

[KSZ92] S. Khanna, M. Sebrée, and J. Zolnovsky. Realtime scheduling in SunOS 5.0. In *Proceedings of the USENIX Winter Conference*, pages 375–390, 1992.

[LL73] C.L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[LMKQ89] 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, May 1989.

[Mil92] D.L. Mills. Network time protocol (version 3): Specification, implementation, and analysis. *Network Working Group Request for Comments 1305 (RFC 1305)*, March 1992.

[NYM91] J. Nakajima, M. Yazaki, and H. Matsumoto. Multimedia/realtime extensions for the Mach operating system. In *Proceedings of the Summer 1991 USENIX Conference*, pages 183–198, Nashville, Tennessee, June 1991.

[PP91] C. Partridge and S. Pink. An implementation of the Revised Internet Stream Protocol (ST-2). In *Proceedings of the 3rd MultiG workshop*, Stockholm, Dec. 1991.

[TNR90] H. Tokuda, T. Nakajima, and P. Rao. Real-time Mach: Towards a predictable real-time system. In *Proceedings of USENIX Mach Workshop*, October 1990.

[Top90] C. Topolcic. Experimental Internet Stream Protocol, version 2 (ST-II). *Network Working Group Request for Comments 1190 (RFC 1190)*, Oct. 1990.

Acknowledgements

We wish to thank Bengt Ahlgren and Kjersti Moldeklev for suggestions and comments on this paper, and for patiently providing "normal background activity" on our experiment machines. We also want to thank Fredrik Orava and Stephen Pink for their constructive comments on this paper.

Biographies