Predictability Requirements of a Soft Modem

Michael B. Jones
Microsoft Research, Microsoft Corporation
One Microsoft Way
Redmond, WA 98052, USA
mbj@microsoft.com
http://research.microsoft.com/~mbj/

Stefan Saroiu
Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350, USA
tzoompy@cs.washington.edu
http://www.cs.washington.edu/homes/tzoompy/

ABSTRACT
Soft Modems use the main processor to execute modem functions traditionally performed by hardware on the modem card. To function correctly, soft modems require that ongoing signal processing computations be performed on the host CPU in a timely manner. Thus, signal processing is a commonly occurring background real-time application—one running on systems that were not designed to support predictable real-time execution. This paper presents a detailed study of the performance characteristics and resource requirements of a popular soft modem. Understanding these requirements should inform the efforts of those designing and building operating systems needing to support soft modems. Furthermore, we believe that the conclusions of this study also apply to other existing and upcoming soft devices, such as soft Digital Subscriber Line (DSL) cards. We conclude that (1) signal processing in an interrupt handler is not only unnecessary but also detrimental to the predictability of other computations in the system and (2) a real-time scheduler can provide predictability for the soft modem while minimizing its impact on other computations in the system.

Keywords
Real-Time, CPU Scheduling, Soft Devices, Soft Modem, Signal Processing, Open Real-Time System, Windows NT, Windows 2000, Rialto, Rialto/NT.

1. INTRODUCTION
Soft modems use the main processor to execute modem functions traditionally performed by the digital signal processor (DSP) on the modem card. Soft modems have enjoyed large success in the home computer market. Two reasons for their success are low cost and the flexibility of migrating to newer technologies by simple software upgrade. Given recent advances in CPU processing power, the impact of a soft modem on the throughput of the system is reasonable—we measured a 14.7% sustained CPU load on a 450 MHz Pentium II. Because soft modems need periodic real-time computations on the host CPU in order to maintain line connection and transmit data, a mechanism ensuring predictable scheduling is essential.

Appeared in *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Cambridge, MA, June 2001.

This paper presents a detailed study of the performance characteristics and resource requirements of a popular soft modem[1]. We analyzed the vendor-supplied version of the driver and three additional versions that we created. The four versions of the soft modem driver are:

- *INT* – the signal processing routines are executed in an interrupt handler (a.k.a. in *interrupt context*). This is the original version.
- *DPC* – the signal processing routines are executed in the context of a Deferred Procedure Call (DPC) [21, pages 107-111].
- *THR* – the signal processing routines are executed in the context of a kernel thread scheduled using the standard priority-based Windows 2000 scheduler.
- *RES* – the signal processing routines are executed in the context of a kernel thread scheduled using a Rialto/NT CPU Reservation [16].

We captured performance profiles of each of the four versions of the driver and compared and contrasted the modem performance. We report on the benefits and the problems encountered with each of the driver versions analyzed. One of the goals of this study is to make the detailed performance characteristics of a popular soft modem available to the industry, allowing this data to inform their work on providing predictable execution on consumer and general-purpose operating systems.

While the soft modem's 14.7% CPU load is not high *per se*, a problem with the vendor version is that all of this time is spent in interrupt context. Once connected, the execution of the interrupt handler typically lasts 1.8ms with a repeatable worst case of 3.3ms during connection.

This study shows that signal processing in interrupt context is not only unnecessary but also detrimental to the predictability of other computations in the system. While DPCs and priority-based scheduling cause milder side effects upon the rest of the system, they nevertheless suffer from some of the same drawbacks as the original version. We conclude that certain kinds of real-time scheduling abstractions provide a better answer to the observed predictability problems that can be caused by the soft modem.

Indeed, their real-time requirements and omnipresence make the soft modems an excellent environment for testing the feasibility and the practicality of soft real-time scheduling for commodity operating systems. One section of our results is dedicated specifi-

[1] Our agreement with the manufacturer prevents us from identifying this soft modem.

cally to studying the effectiveness of using a particular set of real-time scheduling abstractions for supporting the predictability requirements of the soft modem. This paper demonstrates a concrete set of benefits when real-time scheduling is applied to the computations employed by a soft modem.

Finally, we believe many of the lessons learned from studying soft modems are applicable to a wider class of problems. Other soft devices are already in use, with more coming soon. For instance, software-based *Digital Subscriber Line* (Soft DSL) [22] devices have been announced. As will be detailed, Soft DSL has similar execution period requirements, but significantly larger overall CPU requirements than soft modems.

The remainder of this paper is structured as follows: Section 2 provides background on soft modems and operating systems support for predictable execution. Section 3 describes the hardware and software tools used for our study. Section 4 details how the soft modem used in these experiments operates. Section 5 discusses the four soft modem driver versions used in this study. Section 6 contains our results. A roadmap of the results can be found at the beginning of Section 6. Section 7 discusses possibilities for further related research. Section 8 relates some industry perspectives on the findings of the study. And Section 9 presents our conclusions.

2. BACKGROUND

2.1 Modem Taxonomy

A modem is a peripheral device that enables computers to communicate with each other over conventional phone lines. The term modem stands for *Modulator/Demodulator*. The purpose of a modem is to convert (modulate) the *digital* signal that a computer understands into an *analog* signal that can be carried over a phone line, and to re-convert (demodulate) the analog into a digital signal at the other end [3]. Demodulation consists of digitizing analog waveforms using an *A-to-D* converter followed by the application of *signal processing* algorithms. Modulation consists of a different set of signal processing algorithms to produce a digitized waveform, which is sent through a *D-to-A* converter.

Traditional modem communication standards assume that both ends of a data connection are linked to the public switched telephony network (PSTN) by analog lines. This limits the communication bandwidth to 33.6Kbps in each direction [1]. By assuming that one of the endpoints is connected digitally to the PSTN (like most Internet Service Providers are), modern modems are able to achieve speeds of up to 56Kbps downstream and 33.6Kbps upstream using the V.90 protocol [12].

There are four functions that a modem provides:

1. An interface between analog phone lines and digital computer components – A/D and D/A.
2. Signal modulations at different rates.
3. An Attention (AT) command set interpreter.
4. An asynchronous interface between the modem and the computer.

In addition to these functions, any modem card will provide buffering for data flowing in both directions.

Modems can be classified into hardware-based modems (traditional modems) or software-based modems, depending on where each of these functions are executed.

2.1.1 Hardware-based Modems

Traditional modems implement all the modem functionality in hardware on the modem card. Dedicated chips provide signal modulation and interpret the AT command set. The card also provides A/D and D/A converters. On older modems, the *Universal Asynchronous Receiver/Transmitter* (UART) chip implements the asynchronous interface between the modem and the computer. Today, the PCI bus interface often provides this functionality, replacing the UART chip.

2.1.2 Software-based Modems

Software-based analog modems use the host processor to perform some of the modem functions traditionally performed on the modem card. Two types of software-based modems have emerged [7]:

1. *Controllerless modems* (also known as *winmodems* or *linmodems*), which perform and interpret the standard attention (AT) commands on the main processor. Signal modulation, A/D, and D/A are implemented by hardware on the modem card.
2. *Soft modems*, which perform signal processing (as well as AT commands) on the host CPU, unlike both regular and controllerless modems. Modem data buffers may reside in host memory. Soft modems still have hardware-based A/D and D/A converters.

Today, the software-based analog modems are very common on the new computer systems, both for workstations and especially for laptops. Some of the reasons for their success are low cost, low power consumption, and maximum upgrade flexibility. Drawbacks are high CPU and memory usage. The scant availability of drivers for operating systems other than Windows has also contributed to their limited use on non-Windows platforms.

The soft modem driver that we used for our experiments follows the Windows Driver Model (WDM) standard [21, 19]. Since the Rialto/NT abstractions were implemented on Windows 2000, our study is focused on this operating system.

2.2 Commodity Operating Systems and Real-Time Applications

General-purpose operating systems such as Windows 2000, Linux, and Solaris are increasingly being used to run time-dependent tasks such as audio and video processing despite good arguments against doing so [20]. This is the case even though many such systems, and Windows 2000 in particular, were designed primarily to maximize aggregate throughput and to achieve approximately fair sharing of resources, rather than to provide low-latency response to events, predictable time-based scheduling, or explicit resource allocation. Nonetheless, since these systems are being used for time-dependent tasks, it is important to understand both their capabilities and limitations for such applications.

One common mechanism provided for real-time applications is to designate a range of high priority levels as *real-time priorities*.

For instance, Windows 2000 supports 32 priority levels in three classes:

- *Idle:* Priority 0 is used by the per-processor idle threads.
- *Regular:* Priorities 1-15 are variable levels; thread priorities in this range are adjusted by the system within the range to increase responsiveness. For example, quantum size is increased for threads in the foreground process, priority may be boosted upon completing a wait, and priority is boosted for threads that have been ready to run, but not scheduled for several seconds.
- *Real-Time:* Priorities 16-31 are real-time priorities. Quanta and priorities of threads in this range are not adjusted—the scheduler simply runs the threads at the highest priority in a round-robin manner.

2.3 The Rialto/NT Approach

The CPU Reservation abstraction was developed within the Rialto real-time operating system [14, 13]. This abstraction allows activities to obtain minimum guaranteed execution rates with application-specified reservation granularities. This abstraction was subsequently ported to a research version of Windows 2000 called Rialto/NT [16].

Rialto/NT was designed and built to combine the benefits of today's commodity operating systems with the predictability of the best soft real-time systems. Rialto/NT supports simultaneous execution of independent real-time and non-real-time applications. These goals are achieved by computing a deterministic schedule that meets the declared requirements of all admitted real-time tasks whenever the set of real-time applications changes.

3. ENVIRONMENT AND METHODOLOGY

3.1 Hardware Environment

All performance results reported were measured on a Dell Precision 610 system with a 450 MHz Pentium II, 384 MB ECC SDRAM and a Quantum Viking II SCSI hard drive. The soft modem supports theoretical speeds of up to 56Kbps downstream and 33.6Kbps upstream and a plethora of modem standards including V.90, V.42bis, V.42, and V.34 [12]. The minimum system requirements for this soft modem are:

- 150 MHz Pentium processor or 233 MHz AMD K6/K6-2 processor or 266 MHz Cyrix 6x86 MX processor.
- Windows 95/98 with 16 MB of RAM or Windows NT 4.0 with 32 MB of RAM.
- 2 MB of free disk space.

For all traces (except for results in Sections 6.5.2 and 6.6), we connected to the Microsoft internal network via Remote Access Service (RAS) using the Point-to-Point Protocol (PPP). We used the same phone number that Microsoft employees use to connect from home. The Microsoft RAS Servers use 3Com *Total Control* [2] remote access devices. Under normal conditions, the modem connected at speeds of 50.6Kbps downstream (and occasionally higher) and 31.2Kbps upstream. The Microsoft internal network is a 100Mbps switched network.

For the results presented in Sections 6.5.2 and 6.6, we used a dedicated Microsoft Research RAS Server with a Digi *DataFire* RAS 48 PT2 [8] remote access concentrator device. The modem connected at a downstream speed of 50.6Kbps and an upstream speed of 28.8Kbps.

3.2 Software Environment

3.2.1 Instrumented Windows 2000 Kernel

We used an instrumented version of the Windows 2000 kernel in order to understand and tune the behavior of the system and applications. The kernel is capable of logging a wide variety of events to a physical memory buffer and then dumping them to disk for post-processing. During our experiments, we used predefined instrumentation points to log all deferred procedure calls (DPCs), thread context switches, thread and process creations and deletions, and synchronization events. We also logged application-specific data such as modem hardware register values and modem phase change events.

The instrumented kernel offers the same performance as a regular kernel when no events are to be logged. Furthermore, logging an event has minimal impact on the system performance. We measured an average execution of 247 cycles for logging an event, which translates to about 549ns on the computer used to collect the data.

Logging produced around 10MB of data per minute. After dumping the binary event logs to disk and converting them into a text format, we post-processed the output with Perl scripts that filtered out uninteresting data and converted the remainder into a more readable format.

All the experiments used a kernel that contains both the regular Windows 2000 and the Rialto/NT [16] schedulers. A thread is scheduled based on its priority unless it makes a CPU Reservation via a system call. A thread to which a CPU Reservation is guaranteed is scheduled by the Rialto/NT scheduler.

3.2.2 Soft Modem Driver Source Code

For the soft modem driver, we had access to source code that negotiates the connection, services the card interrupts, and makes the appropriate calls to the signal processing routines. We did not have source code for any signal processing related modem functionality.

The lack of complete source code of the driver did not impede us from studying the predictability of the soft modem. In the vendor version of the driver, the calls to the signal processing routines are made in the interrupt handler. The available source code allowed us to instead make the calls in the context of either a DPC or a thread, as needed in our experiments. We were unable, however, to draw any conclusions about the effectiveness of the signal processing algorithm, nor fully understand or modify its behavior when invoked later than it would have been in the INT version.

4. SOFT MODEM OPERATION

The soft modem uses Direct Memory Access (DMA) to transfer data between memory and the A/D and D/A. Sixteen-bit samples are transferred at rates between 7.2 KHz and 16 KHz. When receiving, whenever a predefined amount of data has been sampled off the phone line, the modem interrupts the system. The interrupt handler processes both incoming and outgoing data. This soft modem uses floating point but does not use MMX instructions for its signal processing algorithms. The driver software must consume incoming and provide outgoing samples without overflow-

ing or underflowing the buffers. There are four different buffers—two output buffers (one for data and one for voice samples) and two input buffers. Each buffer has a size of 512 16-bit samples, for a total of 1024 bytes. Since modems are mainly used for data communication, our experiments traced the data buffers only.

When dial-up is initiated, but before the dial tone, the modem is in an on-hook state performing ring detection. During this period, which lasts about two seconds, interrupts occur whenever 18 samples have been transferred by the DMA to the memory; at a DMA rate of 7.2 KHz, this corresponds to an interrupt rate (the inter-arrival time between interrupts) of 2.5ms. During dialing and initial modem connection attempts, interrupts occur for every 90 bytes of transferred data, corresponding to an interrupt rate of 12.5ms.

After dialing is finished, when the modem attempts to connect, there is an initial period of training during which the modem listens to the phone line trying to determine whether any modem protocols are in use and whether analog to digital conversion is taking place. If no conversion is occurring, the V.90 protocol is used and the modem connects at 56Kbps or less. If there is A/D conversion then the ISP is not connected digitally to the PSTN and the modem uses the V.34 protocol, with its inherent 33.6Kbps connection speed [1].

5. SOFT MODEM DRIVER VERSIONS

5.1 Initial Interrupt-Based Version (INT)

In the initial (vendor-supplied) version of the driver, when the card interrupts the CPU, the driver software performs signal processing inside the interrupt handler (a.k.a. Interrupt Service Routine or ISR). Both outgoing and incoming samples are processed during each interrupt. The handler also services modem requests for changing the transfer frequency and the number of samples per interrupt, both of which, in effect, determine connection speed.

Under Windows 2000, the interrupts are serviced in a priority order based on their *interrupt request levels* (IRQLs). Thus, the modem interrupt handler can be preempted by other interrupt handlers. On our test machine, all other interrupts had higher priority with two exceptions—the interrupts associated with the network card and the SCSI controller for the CD-ROM. Some of the higher priority interrupts include the interrupts servicing the keyboard, the communications ports, the mouse, the audio drivers, the floppy disk and the SCSI hard disk.

5.2 DPC-Based Version (DPC)

Deferred Procedure Calls (DPCs) are routines executed within the kernel in no particular thread context in response to queued requests for their execution. For example, DPCs check the timer queues for expired timers and process the completion of I/O requests. Having interrupt handlers queue DPCs to finish work associated with them reduces hardware interrupt latency. All queued DPCs are executed whenever a thread is selected for execution just prior to starting the selected thread. There cannot be more than one instance of the same DPC inside the queue at any one moment. While good for interrupt latencies, DPCs can be bad for thread scheduling latencies, as they can potentially result in an unbounded amount of work before a thread is scheduled.

We created a version of the soft modem driver that executes the signal processing code in a DPC. When the modem card raises an interrupt, the ISR queues a DPC to process the buffer of samples. Unlike the vendor version, where only higher priority interrupts preempt the signal processing routines, signal processing executing inside of a DPC is preempted by all hardware interrupts.

There can be more than one occurrence of the interrupt before the DPC is executed. Therefore, synchronization between the interrupt handler and the DPC was implemented using an atomically incremented variable that was set to the number of interrupt occurrences. This variable is atomically decremented by each processing unit executed by the DPC.

5.3 Thread-Based Version (THR)

In the THR version of the driver, signal processing is performed in a thread running at a specified priority. The thread is created during driver initialization.

A semaphore was chosen as the synchronization mechanism between the thread and the interrupt handler. Under the Windows 2000 model, a semaphore cannot be directly set from an interrupt handler. Thus, whenever the interrupt occurs, the interrupt handler queues a DPC that signals the thread via the semaphore. As before, since there can be more than one occurrence of the interrupt before the DPC is executed, an atomically set variable is shared between the ISR and the DPC.

Because interrupt handlers and DPCs run to completion before a thread is dispatched, there is a potentially unbounded delay between the interrupt and when the thread starts to run. However, for specific hardware and driver combinations, reasonable delays are achievable in practice [6].

5.4 CPU Reservation-Based Version (RES)

In the final version of the driver, the signal processing thread uses the Rialto/NT real-time scheduler's CPU Reservation abstraction to ensure a minimum guaranteed execution rate and granularity.

CPU Reservation requests are of the form *reserve X units of time out of every Y units for thread A*. This requests that for every time interval of size *Y*, thread *A* be scheduled for at least *X* time units, provided it is runnable. For example, a thread might request 800µs every 5ms, 7.5ms every 33.3ms, or one second every minute.

CPU Reservations are *continuously guaranteed*. If *A* has a reservation for *X* time units out of every *Y*, then for *every* time *T*, *A* will be run for at least *X* time units in the interval [*T*, *T+Y*], provided it is runnable. Execution time intervals granted to a thread for its reservation are not guaranteed to be contiguous. If a thread is not runnable during its reserved time intervals, the time is returned to the Windows 2000 scheduler and used for other threads.

The current implementation of Rialto/NT has two restrictions: (1) CPU reservations must have values that are integer multiples of milliseconds, since they are driven off the periodic Windows 2000 clock and (2) the period of a reservation must be a power-of-two multiple of a millisecond due to a choice of algorithms within Rialto/NT.

Since processors run at different speeds, a CPU Reservation request should have different values on different machines. In practice, developers will typically use one of two adaptive techniques to determine an appropriate CPU Reservation amount: The pro-

gram can dynamically measure the amount of time actually used for its time sensitive computation, and base its reservation amount on the observed runtimes. And it can adjust its reservation amount according to whether the computation was able to finish on time during previous iterations. Both techniques may also be employed in combination.

We analyzed the soft modem under different reservations—1 millisecond every 8 milliseconds (12.5%), 2ms/8ms (25%), 2ms/16ms (12.5%), 3ms/16ms (18.75%), 1ms/4ms (25%). The reservation is requested only after the on-hook phase of about two seconds in order to avoid interference with this high interrupt rate period. During this initial phase, the thread runs without a reservation but with a real-time priority. Since the soft modem causes audio computations during dial-up (different protocol attempts can be heard while connecting), and the kernel audio mixer threads run at priority no higher than 24, we assigned priority 24 to the thread during this period. Having sub-millisecond reservation accuracy would have allowed us to use a different reservation for the on-hook phase, for instance 0.1ms every 2.5ms.

6. RESULTS

Our results are organized as follows: Section 6.1 gives an overview of our experiments. Section 6.2 quantifies the resource usage and timing behavior of the original soft modem driver, plus the DPC and THR versions. Section 6.3 measures the drivers' interference with other applications' operation. Section 6.4 describes the behavior of the soft modem and its effects upon other applications when CPU Reservations are used to schedule its signal processing computations. Section 6.5 quantifies the effects that the different implementations have upon end-to-end modem download throughput. Section 6.6 attempts to precisely characterize the minimum resource and timing requirements that the soft modem must obtain in order to function correctly.

6.1 Overview of Experiments

We analyzed the behavior of the modem under two main scenarios—dial-up and steady state communication. These are the experiments we ran:

- Establishing a connection with and without contention by a synthetic CPU load. The CPU load we applied was a normal priority (priority 8) spinning thread.
- A highly compressed file transfer with and without a normal priority spinning thread.
- Different real-time and normal thread priorities for the THR version, with a spinning competitor.
- Different reservation amounts for the RES version.
- A stress scenario where the soft modem driver, a normal priority (priority 8) spinning thread and a priority 10 process performing a *grep* command that searches the entire file system were run concurrently. By running the *grep* over the entire file system, we ensured disk activity. We found the stress scenario results to be identical to those for a connection with a normal priority competitor. The induced disk activity did not influence the soft modem driver behavior.

In addition to these tests, we studied the impact of the different driver versions on the scheduling latency observed by a real-time thread that uses Windows multimedia timers to request a callback every 1ms.

In order to quantify the observed modem behavior, we measured the following parameters:

- The times between successive soft modem interrupts.
- The times between successive DPC calls and thread wake-ups.
- The elapsed times spent in the soft modem interrupt handler, DPC, and thread. (Note that these times include processing times spent in other contexts that preempt the traced ISR, DPC, or thread. We believe that this is appropriate, since signal processing times are the variable of interest and not execution times.)
- The number of samples pending to be processed. This variable directly reflects whether the modem is meeting its deadlines and whether it recovers from temporary unprocessed data accumulations. This value will always be less than the buffer size of 512 samples since it will wrap around to zero (indicating loss of data) should it ever reach 512.
- The modem's effect on the scheduling latency of coexisting threads.
- The end-to-end modem throughput. This is a primary measure of user-visible modem behavior.

Whenever possible, we compared the measured soft modem performance to the *PC 99* specification recommendations [11]. *PC 99* was created by Intel and Microsoft as a set of recommendations to hardware manufacturers and driver writers for the Microsoft Windows family of operating systems.

6.2 Soft Modem Resource Usage Study

The measurements presented in this section use a test scenario of a dial-up connection with a normal priority spinning competitor thread.

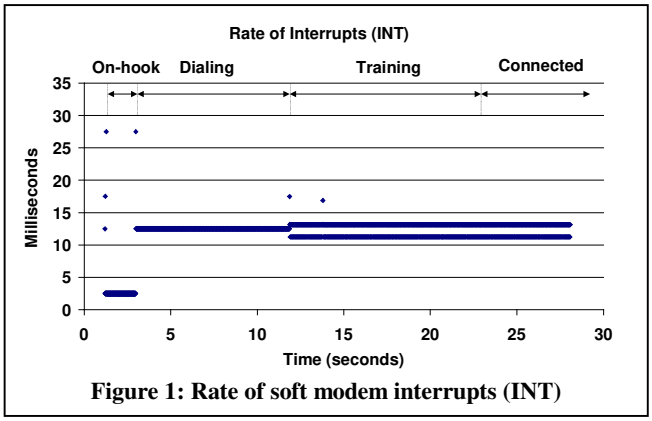

Figure 1: Rate of soft modem interrupts (INT)

6.2.1 Soft Modem Interrupt Rate

Figure 1 shows the interrupt rate for the INT version. Before dialing, interrupts occur very frequently (every 2.5ms) for about two seconds—the modem is on-hook performing ring detection. For the next 9 seconds, interrupts occur every 12.5ms while the modem is dialing and waiting for the other end to start the connection. Whenever there is a change in DMA frequency or in the size of the sample buffers, the modem requests an interrupt frequency change. This request causes a short delay in the interrupt rate that corresponds to the six scattered points in the graph. Once the connection has been established, interrupts occur every 13.125 or 11.25 milliseconds. The rates fall within the *PC 99* recommended

interrupt rates of 3-16 milliseconds [11]. The other driver versions have the same interrupt rates.

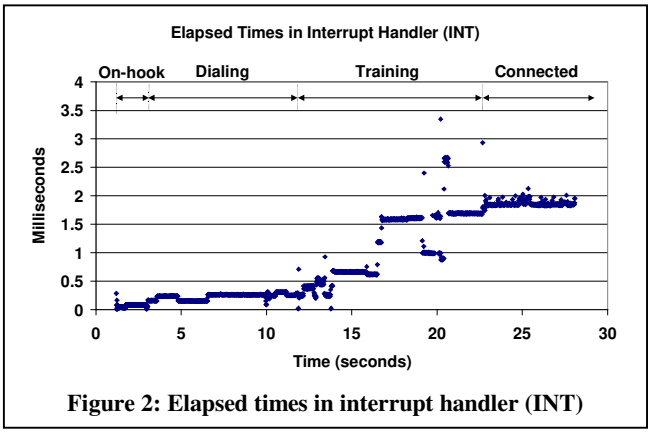

Figure 2: Elapsed times in interrupt handler (INT)

6.2.2 *Elapsed Times in ISR in INT Version*

The *PC 99* specification recommends that the maximum time during which a driver-based modem disables interrupts should not exceed 100µs [11]. Figure 2 shows that the execution of the interrupt handler typically lasts 1.8ms with a repeatable worst case of 3.3ms—a factor of 33 worse than the specs recommend. The elapsed times measure the times needed to handle the soft modem interrupts and include time spent in other interrupts that might have preempted the soft modem interrupt handler.

We believe that spending this much time in interrupt context has unacceptable consequences for the predictability of coexisting real-time activities. For instance, multimedia timers in Windows 2000 allow applications to schedule callbacks at millisecond granularity. Obviously, this resolution will not be attainable if any ISRs run for longer than 1ms. We quantify multimedia timer delay in Section 6.3.

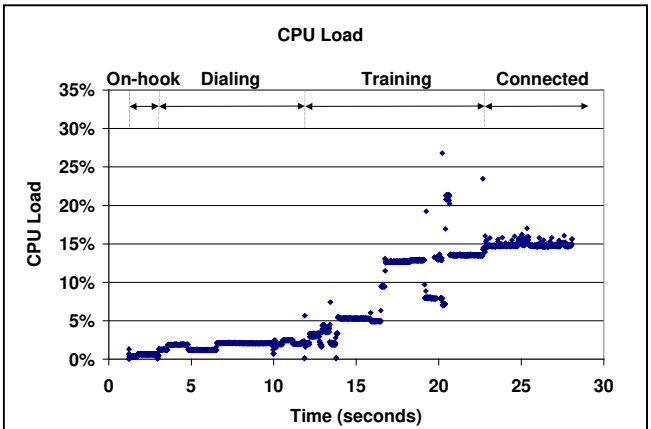

Figure 3: CPU utilization (INT)

6.2.3 *CPU Utilization*

Figure 3 shows the CPU utilization of the soft modem. Each point represents the utilization during a 12.5ms interval while the soft modem code was executing. We chose 12.5ms because this is the average period between soft modem interrupts during all but the first two seconds of a connection. The connection is established after 23 seconds, thus the last five seconds of the trace present the CPU load in steady state. As Figure 3 illustrates, the soft modem uses 14.7% of the CPU once connected.

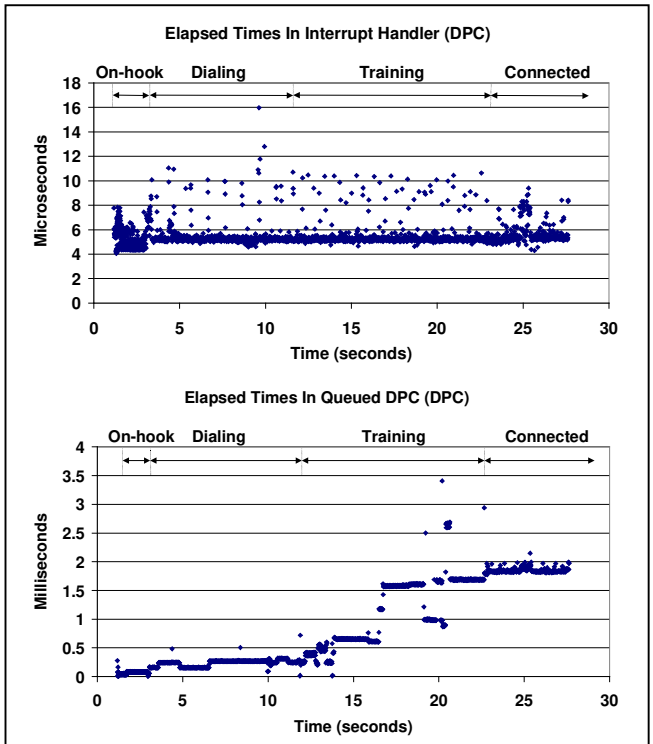

Figure 4: Elapsed times in interrupt handler (above) and in queued DPC (below) (DPC)

6.2.4 *Times Spent by the DPC Version*

When signal processing is done in a DPC, the interrupt handler does very little work—an average of 5.4µs per interrupt and a maximum of 16µs. Figure 4 shows the times spent inside of the interrupt handler and the DPC. Unsurprisingly, the times spent in the DPC are essentially identical to those spent in the ISR in the original version in Figure 2.

While the ISR execution times have been reduced from milliseconds to a few microseconds, the time spent inside the DPC is still too large. The *PC 99* specifications suggest that at any instant in time, the total execution time required for all DPCs that have been queued by a driver-based modem, but have not been executed, should not exceed 500µs [11].

6.2.5 *Samples Pending to be Processed*

Executing signal processing in a thread context has the benefit of minimizing the times spent in interrupt handlers and DPCs. While the predictability gains are obvious, the question becomes whether the soft modem is able to process data and maintain the line connection. A good indicator of the connection performance is the number of samples pending to be processed in the receive data buffer. Figure 5 shows these unprocessed samples for the vendor version, measured after the call to the signal processing routine returns.

In the training and connected phases, there are never more than 30 samples left unprocessed in the buffers, which is very small relative to the size of the buffer (512 samples). The DPC version has approximately the same number of unprocessed samples.

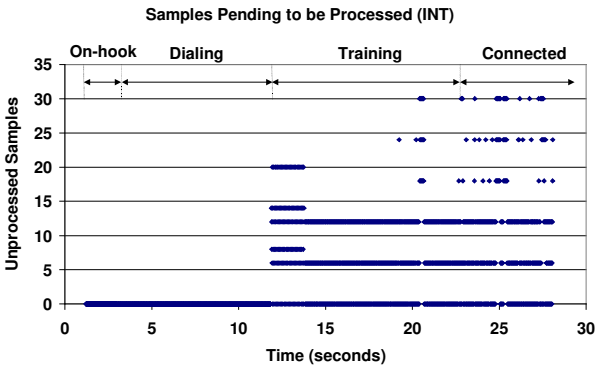

Figure 5: Samples pending to be processed with a normal priority spinning thread (INT)

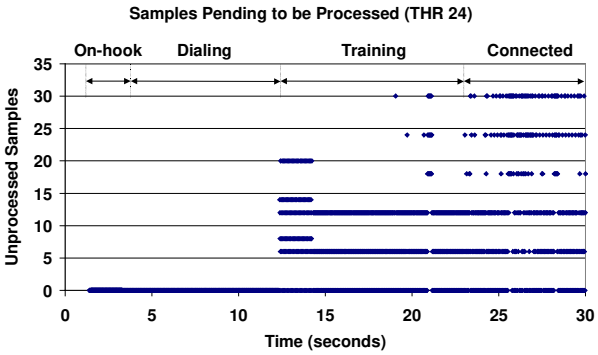

Figure 6: Samples pending to be processed with a normal priority spinning thread (THR 24)

Figure 6 shows the samples pending to be processed for the THR driver version, measured after the call to the signal processing routine returns. The *PC 99* specs recommend that drivers should perform long computations in a real-time thread using priorities 28 and above. After trying different priority settings, we concluded that priority 24 suffices, which is the same as the thread priorities of the audio drivers that are used by the soft modem to output the modem noises during connection.

In Figure 6, the THR version of the driver is able to keep up with the received samples, even in the presence of normal priority competition. There are more cases when there are 30 samples left unprocessed, but overall, the behavior looks very similar to the vendor version.

6.2.6 Samples Pending for a Failed Connection

In order to understand the modem behavior under severe competition, we lowered the priority of the signal processing thread. Figure 7 shows the samples pending to be processed when the processing thread has normal priority (priority 8) and there is a normal priority spinning competitor. The soft modem is not able to dial the number properly. Instead, the *"Please hang up and try your call again"* message is heard. As Figure 7 illustrates, running the modem thread at too low a priority causes buffer overflows when competition is present.

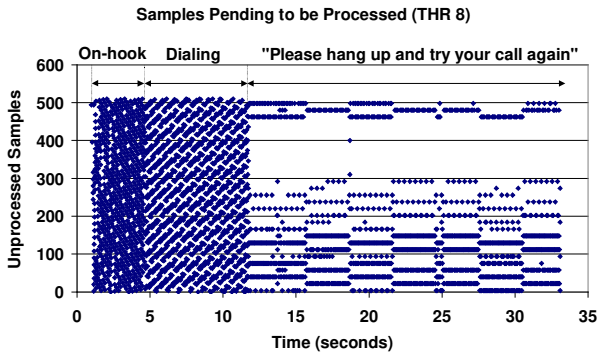

Figure 7: Samples pending to be processed with a normal priority spinning thread (THR 8)

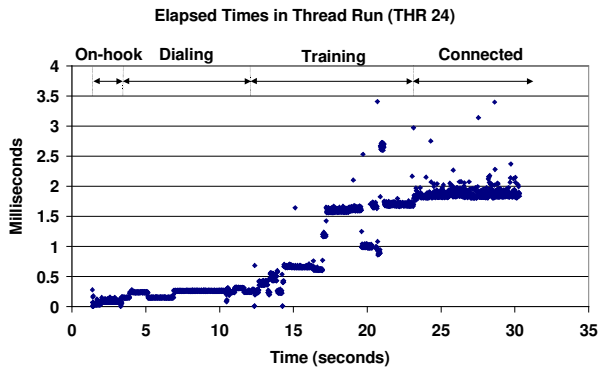

Figure 8: Elapsed times in thread run (THR 24)

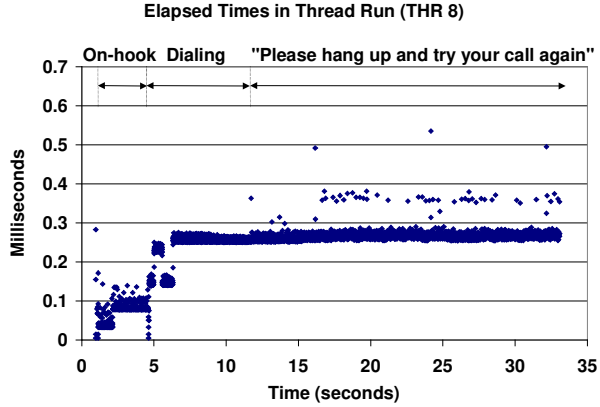

Figure 9: Elapsed times in thread run (THR 8)

6.2.7 Elapsed Times in the Signal Processing Thread

Figure 8 shows the elapsed times spent per thread wakeup for the THR driver version at priority 24 with a normal priority spinning competitor.

The elapsed times for the thread runs look identical to the elapsed times in the interrupt handler for the vendor version and the elapsed times in a DPC for that version. This is expected, since there is no real-time competitor for the signal processing thread.

As Section 6.2.6 shows, when the signal processing thread has priority 8, the modem is not able to dial the number properly un-

der the presence of a normal priority spinning competitor. Since no data is exchanged over the phone lines, there is very little time elapsed per thread run for signal processing, as seen in Figure 9.

6.3 Interference with Scheduling Predictability of Other Applications

In order to understand the effects of long running ISRs and DPCs, we measured the wakeup latencies of a callback routine that uses Windows multimedia timers. The timers have been set to fire every millisecond and the routine is called with priority 31, the highest priority for a real-time thread.

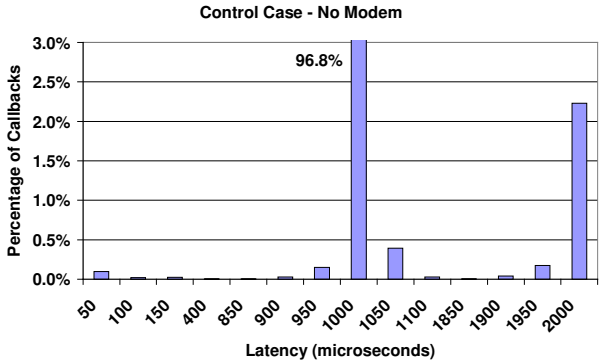

Figure 10: Latency histogram for a 1ms callback routine without the soft modem

6.3.1 Control Case: Timer Latency with no Modem

Figure 10 shows a histogram of the measured timer wakeup latencies on a quiescent machine when the soft modem is not running. The experiment captured 30,000 wakeups over a 30-second period. Samples are accumulated into 50µs buckets. On this machine, timer wakeups are triggered by the *Real-Time Clock* (RTC), which is interrupting at a rate of 1024Hz, or every 976µs. (It supports only power-of-two frequencies.) Thus, as described in [15], approximately 2.4% of the wakeups occur near 2ms, since clock interrupts arrive 2.4% faster than the desired 1ms timer wakeup rate.

6.3.2 Timer Latency with INT and DPC Versions

Figures 11 and 12 show histograms of the measured latencies when the soft modem is added, for the INT version and for the DPC versions, respectively. The damage the soft modem's long-running ISR or DPC causes to the predictability of the callback routine is evident: the tails of the distributions increase from 2ms to over 5ms. This is precisely the reason why the *PC 99* modem guidelines recommend that such long-running computations be performed in threads.

6.3.3 Timer Latency with THR Version

Figure 13 illustrates the callback latencies for the THR driver version. As before, it uses a priority 24 real-time thread. By running the modem computations in a thread, timer wakeup latencies are once again nearly as predictable as those for the control case in Figure 10.

Given that thread-based signal processing works well and causes less disruption of coexisting system activities, why then might a vendor still chose to perform signal processing in DPCs or in interrupt handlers instead of a thread?

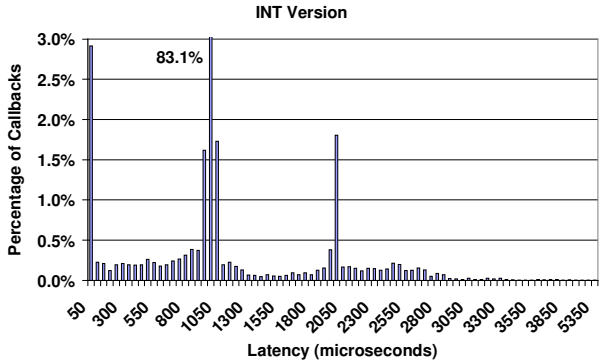

Figure 11: Latency histogram for a 1ms callback routine with the vendor driver version (INT)

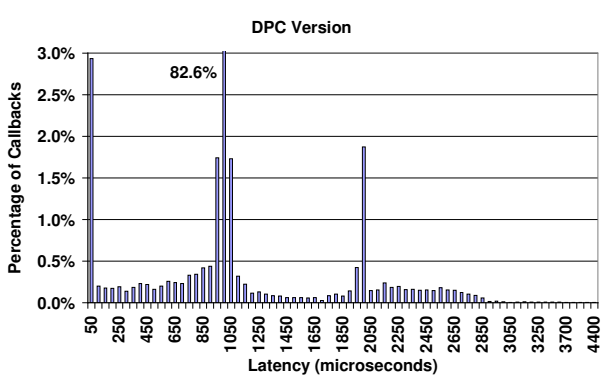

Figure 12: Latency histogram for a 1ms callback routine with the DPC version

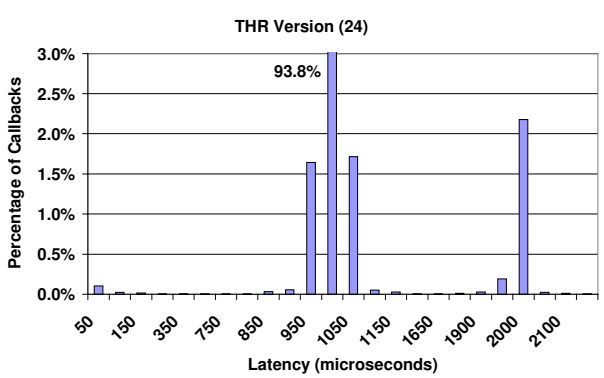

Figure 13: Latency histogram for a 1ms callback routine – THR version (THR 24)

6.3.4 Reflections upon the Vendor Choice

With threads, vendors face a problem common to all priority-based open systems (ones in which independently authored applications and/or drivers may be executed): for any chosen priority, there is a potentially unbounded delay until a thread is scheduled to run. These delays can be caused by other applications running for arbitrary periods of time at the chosen or higher priority. Thus, *no timing guarantees can be made.*

For systems with a fixed priority preemptive scheduler, like Windows 2000, one solution would be to use Rate Monotonic Analysis (RMA) [18] to assign priorities such that all time-dependent

tasks can meet their deadlines. (Rate monotonic analysis does two things: it assigns priorities to periodic tasks in order of their periods, with higher priorities going to tasks with shorter periods. And it determines whether the entire resulting schedule is feasible, based upon the resulting priority assignments and computational requirements of each task.) Unfortunately, RMA cannot be practically employed because:

- RMA assumes cooperation between the threads, which is unrealistic in an open system, given the existence of independently developed drivers and applications written by different vendors running together on the same operating system.
- RMA assumes constant timing requirements for all the coexisting threads. Whenever the CPU requirements of a thread change, it potentially affects the scheduling all the other coexisting threads.

We believe that a better alternative to RMA in an open system would be a real-time scheduler such as Rialto/NT. The coexisting threads could then reserve ongoing portions of the CPU according to their needs, using the CPU Reservation abstraction. They would be then guaranteed timely execution even in the presence of competition. This alternative is explored in the next section.

6.4 Rialto/NT Real-Time Scheduling Results

This section presents results achieved by scheduling the soft modem's signal processing computations using CPU Reservations provided by the Rialto/NT scheduler. All experiments in this section were run with a normal priority spinning competitor thread.

6.4.1 Samples Pending to be Processed in RES Version

Figure 14 shows the samples pending to be processed for the RES driver version for a 2ms every 8ms CPU Reservation, which reserves 25% of the CPU. Note that there are unprocessed samples left in the buffers, but the modem is able to process them in time and no buffer overflows occur. This situation occurs because a 2ms every 8ms CPU Reservation only approximates the desired 2ms every 12.5ms reservation. While there are more pending unprocessed samples than in the vendor version or THR version without real-time competition (see Figure 6), we believe that this is a small price to pay in exchange for the gains in the predictability of the coexistent system activities. The number of samples pending to be processed is much smaller than the receive buffer size of 512 samples, and there is no degradation in the modem performance.

Furthermore, no matter how many competing threads are introduced at any priority, or with other reservations, the modem's CPU reservation always guarantees it the same amount of time, allowing it to function just as well as the case shown here. Other programs' behaviors will not drive the modem into the "Please hang up and try your call again" state that can happen when using threads without reservations—a key advantage of the RES version over the THR version.

6.4.2 Elapsed Times per Wakeup in RES Version

Figure 15 presents the elapsed times spent in the signal processing thread per thread wakeup. The elapsed times are the times needed for a single run to complete signal processing. As mentioned before, this incorporates times spent in activities that preempt the driver thread.

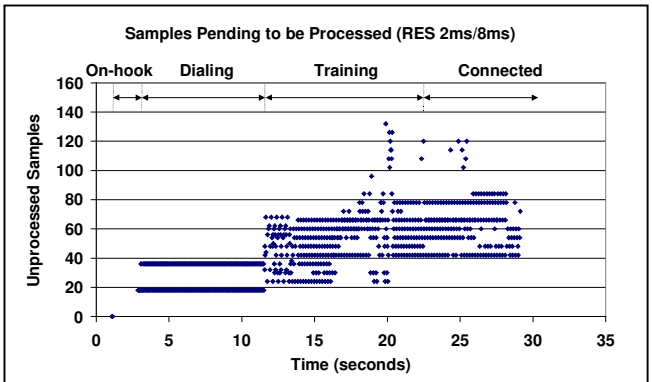

Figure 14: Samples pending with a normal priority spinning thread (RES 2ms/8ms – 25%)

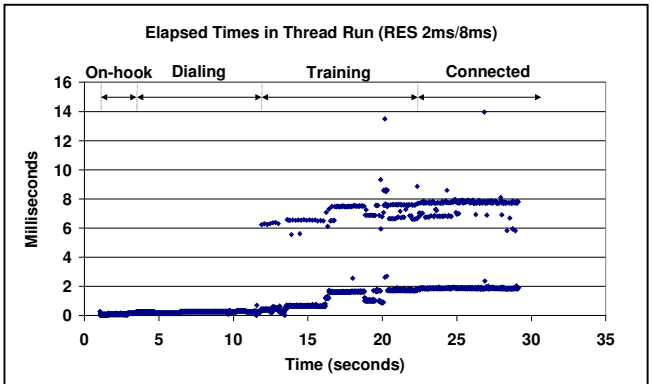

Figure 15: Elapsed times in thread run (RES 2ms/8ms – 25%)

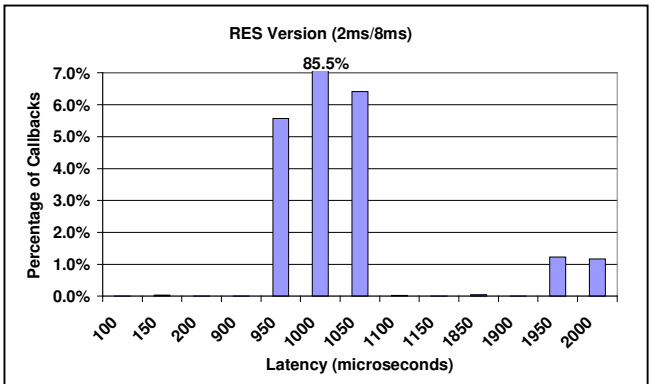

Figure 16: Latency histogram for a 1ms callback routine – RES version (RES 2ms/8ms)

While larger as a percentage than the actual modem requirements, a 2ms/8ms CPU Reservation is not an ideal match for the soft modem processing routines. The period of 8ms causes the signal processing thread to be scheduled to execute at different times than the occurrences of the interrupt. Whenever scheduled, the thread will cede its reservation to the normal spinning competitor if it is not in a ready state. Also, data can be available when the thread is outside its reservation, thus having to compete with the normal priority spinning thread. This behavior is illustrated in Figure 15 by the elapsed times of 6-8ms for signal processing. However, despite the period mismatch, this reservation does allow the modem to operate perfectly, as the results in Section 6.5 show.

6.4.3 Coexistent Thread Latencies in RES Version

Section 6.3 illustrated the impact of the INT and DPC versions of the soft modem on the predictability of a callback routine. Figure 16 shows the callback latencies for the RES version with a 2ms/8ms CPU Reservation.

The predictability of the callback routine improves substantially over the INT and DPC versions. Note that the callback routine is scheduled by the Windows 2000 scheduler with a priority of 31. The predictability is similar to the THR version shown in Figure 13, albeit with three times more callbacks occurring one histogram interval of 50µs to the left or right of the ideal 1ms callback period in this version than in the THR version.

6.5 End-to-End Modem Download Throughput

To analyze modem throughput, we measured the time required to transfer a 200,000 byte file. The file is compressed to defeat modem data compression. We repeated the experiment ten times for each version of the driver.

6.5.1 Microsoft RAS Server Pool

We placed the file on a computer on the Microsoft intranet and then we measured the transfer times, connecting to the Microsoft RAS server, as before. There are two components of the measured times: (1) transfer times of the file through the Microsoft internal network (from the source computer to the RAS server) and (2) transfer times through the phone line (from the RAS server to the destination computer). While including the network times introduces some noise into the modem transfer times, this is a realistic scenario; therefore, we chose to include these results.

	Min	Max	Mean	Std Dev	Succ
INT	37.914	37.258	37.222	0.019	10
DPC	37.151	37.303	37.229	0.051	10
THR Pri 8	59.899	60.658	60.219	0.228	10
THR Pri 24	37.147	40.648	37.560	1.086	10
RES 2ms/16ms	156.632	240.932	204.146	35.447	10
RES 3ms/16ms	37.864	122.042	76.840	30.775	10
RES 1ms/8ms	43.741	83.336	56.237	10.73	9
RES 2ms/8ms	37.086	37.242	37.175	0.053	10
RES 1ms/4ms	37.118	38.823	37.354	0.518	10

Table 1: File transfer times (seconds) of 200,000 bytes including network transfer times

Table 1 contains statistics about the transfer times recorded in seconds, along with the number of successful file copies out of a total of ten attempts. For the THR priority 8 test, there was no spinning competitor; otherwise the modem cannot keep the connection alive for the entire transfer. For all the other tests, a normal priority spinning thread was executing in parallel with the file transfer.

The 2ms every 8ms and 1ms every 4ms reservations (25% CPU) behaved identically to the INT and the DPC version, while the 1ms/8ms (12.5%), 3ms/16ms (18.75%) and 2ms/16ms (12.5%) needed a longer amount of time for transfers.

6.5.2 Microsoft Research Dedicated RAS Server

Next, we eliminated the network transfer times by placing the file on a RAS server itself. We could not do this on the Microsoft RAS servers, since running controlled experiments on the large modem pool would have been infeasible both administratively and technically. Instead, we used a dedicated, Microsoft Research RAS server.

	Min	Max	Mean	Std Dev	Succ
INT	36.334	36.398	36.367	0.029	10
DPC	36.272	36.447	36.396	0.048	10
THR Pri 8	36.533	37.000	36.716	0.152	10
THR Pri 24	36.319	36.475	36.384	0.056	10
RES 2ms/16ms	329.485	363.891	346.688	24.329	2
RES 3ms/16ms	94.615	174.070	103.789	24.735	10
RES 1ms/8ms	N/A	N/A	N/A	N/A	0
RES 2ms/8ms	36.319	36.444	36.378	0.038	10
RES 1ms/4ms	36.303	36.425	36.345	0.036	10

Table 2: File transfer times (seconds) of 200,000 bytes excluding network transfer times

Having eliminated the potential variability introduced by the network, the file transfer times and success rates out of ten attempts are presented in Table 2. As before, there is a spinning competitor thread for all the tests except the THR priority 8 test.

This set of results is similar to the ones presented in the previous section, with two differences:

- The RES driver version using a 1ms/8ms reservation disconnected so frequently it was impossible to run the experiments. Likewise, the 2ms/16ms reservation was nearly unusable.
- On average, the transfer times are both lower and more predictable, due to eliminating the network transfer.

We believe that the disconnections for the 1ms/8ms and 2ms/16ms cases of the dedicated RAS server are most likely caused by the difference in the type of modem at the server. Remember that the corporate RAS server bank uses 3Com *Total Control* [2] remote access devices, whereas the dedicated server uses a Digi *DataFire* RAS 48 PT2 [8] remote access concentrator device. Another difference is that while both servers connected at 50.6Kbps in the downstream direction using the V.90 protocol, the server pool upstream connections occurred at 31.2Kbps, whereas the dedicated upstream connections occurred at 28.8Kbps. All of this serves to illustrate that the modem protocols are complicated, and two standards-compliant implementations may still operate differently.

Nevertheless, while the corporate and research server results are not directly comparable due to the server hardware differences, we have succeeded in providing a more accurate measure of end-to-end modem throughput.

6.6 Reservation Sensitivity Study and Model

As the data above shows, the reservation parameters chosen are critical to modem performance. A sufficient reservation can make the RES version perform as well as the original driver, whereas a poorly chosen reservation can render the modem inoperable. In order to better understand the characteristics of these reservation ranges, and to attempt to develop a predictive model for them, we undertook the following study.

First, we constructed a modified version of the Rialto/NT scheduler that removes the restriction that reservation periods be a power-of-two multiple of a millisecond, instead allowing us to make a single reservation with a period of any integer number of milliseconds. And unlike Rialto/NT, in which a thread remains eligible for timeshare scheduling outside its reserved time slots,

this new scheduler never allows a thread to run outside of its reserved time slots. We made these changes in order to be able to more precisely quantify exactly how much time the soft modem needs to operate correctly.

We then ran a series of controlled experiments, varying the modem reservation parameters, in which we re-measured the file transfer times to the dedicated RAS server, as per the previous section. With the thread forced to live within its reservation by the modified scheduler, we found the results extremely consistent across runs—they tended to either work essentially perfectly or not work at all, depending upon the reservation. Table 3 shows transfer times and success rates out of ten attempts for key sets of reservation values during these experiments.

	Min	*Max*	*Mean*	*Std Dev*	*Succ*
RES 1ms/7ms	36.333	36.724	36.426	0.112	10
RES 1ms/8ms	N/A	N/A	N/A	N/A	0
RES 2ms/13ms	36.288	36.975	36.547	0.232	10
RES 2ms/14ms	38.631	91.713	65.172	37.535	2
RES 2ms/15ms	N/A	N/A	N/A	N/A	0
RES 3ms/15ms	36.275	36.586	36.387	0.108	10
RES 3ms/16ms	97.289	180.415	110.523	26.408	9
RES 3ms/17ms	N/A	N/A	N/A	N/A	0
RES 4ms/16ms	36.255	37.116	36.415	0.256	10
RES 4ms/17ms	N/A	N/A	N/A	N/A	0
RES 7ms/20ms	N/A	N/A	N/A	N/A	0
RES 8ms/20ms	36.347	36.476	36.394	0.039	10

Table 3: File transfer times (seconds) for reservations not also receiving timeshared time

Figure 17 is a graphical representation of the space of possible CPU reservations in which the reservations from Table 3 are plotted, differentiating three cases: reservations that were **Sufficient** for the modem to operate correctly, reservations that were **Insufficient** for the modem to operate correctly, and reservations that were **Marginal**—those for which the modem operated in a degraded fashion. The **Actual** average interrupt amount and period for the INT version of the driver, 1.84ms out of each 12.5ms, is also presented as a point of reference.

Also, two lines that play a role in our model of the soft modem's reservation requirements are included in the graph. One is a line from the origin through the "actual" point (1.84ms every 12.5ms). This slope corresponds to a CPU reservation percentage of 14.7%. The second is a line dividing the space into regions where the reservation period and amount differ by more or less than 12.5ms. In other words, points to the right of this line have reservations with gaps in them longer than 12.5ms.

For all reservations receiving at least 14.7% of the CPU with gaps of under 12.5ms, the soft modem operated perfectly. The observed boundary between working reservations and non-working reservations closely approaches the two boundary lines, with an inflection point near their intersection. Indeed, our model of the modem's operating ranges predicts that all reservations in this region of the space should operate correctly, whereas all reservations outside this region should fail. The intuition behind this model's predictions is explained below.

First, all points below the 14.7% line represent reservations receiving a smaller share of the CPU than the original version actually uses when executing an average of 1.84ms of work every 12.5ms. With insufficient CPU time, the modem eventually falls

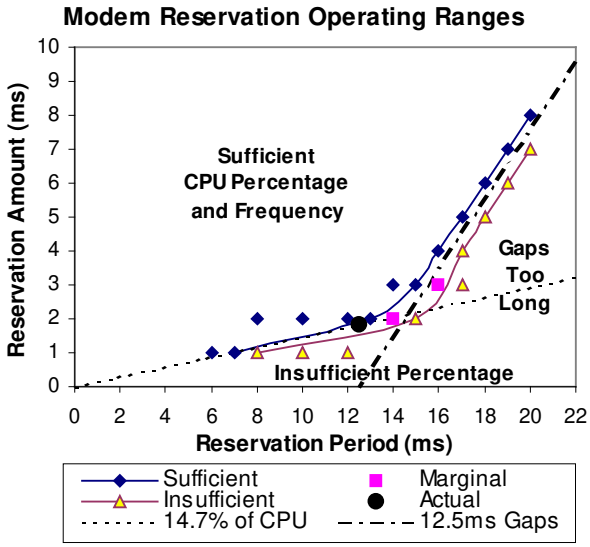

Figure 17: Areas of CPU reservation space where modem does and does not function correctly. "Actual" value is INT version interrupt rate and amount.

behind and loses data. Second, all points to the right of the 12.5ms gap line represent reservations in which there are periods of time longer than 12.5ms during which the modem gets to do no work. During these long gaps, more than one interrupt's worth of work can arrive. Yet despite the RES version calling the signal processing code multiple times when multiple interrupts occur, the code appears to sometimes not successfully process an old interrupt's work once a new one has occurred. (However, without source for the signal processing code, we are unable to verify this assumption.) Apparently the signal processing routine must be called at least every 12.5ms if it is to work correctly.

All insufficient and marginal reservation values fail one or both of these tests—either their percentage is too small, or their gaps are too long, or both—conforming to our model of what kinds of reservations should and should not result in the modem operating successfully. Indeed, the locations of the two marginal results, where the connection sometimes fails or where the transfer rate is significantly degraded, likewise affirm the validity of our model—both are close to, but on the wrong sides of the boundary lines.

We would be remiss, however, if we did not point out that a reservation of 1ms every 7ms proved to be sufficient, even though this is only 14.3% and our model predicts that 14.7% should be required. While very close to the predicted threshold, this point seems to indicate that the behavior in this region is not completely linear.

To summarize, it appears that both a minimum average fraction of the CPU must be delivered *and* it must be delivered frequently enough in order for the soft modem to function correctly. Both the fraction and the frequency were easily determined by observing the behavior of the original interrupt-based driver. The observed data closely fit this predictive model.

Finally, we believe that other real-time tasks, such as Soft DSL, which also involve filling and/or emptying a buffer at a constant rate will be likely to exhibit similar behavior, albeit, with different parameters as dictated by the particular buffer size and rate values.

7. FURTHER RESEARCH

Our study is one step in understanding the application benefits of using real-time schedulers. Soft modems are an ideal platform for prototyping different real-time system abstractions due to their precise timing requirements. One potential extension to our study would be the analysis of the application impact of CPU reservations for multiprocessor machines.

Multiple soft modems serviced by the same driver on a single system would pose different challenges to the real-time schedulers depending on the number of simultaneously communicating modems. Likewise, more studies are needed to understand the overall system behavior when multiple real-time applications using the Rialto/NT scheduling abstractions are concurrently executing. One opportunity would be to conduct experiments in which both the soft modem and the digital audio player application, which was studied in [17], are present.

It would likely prove interesting to construct and study a version of the driver that used an extension to Windows 2000 providing hard real-time timing guarantees, such as RTX [5].

Finally, this research could be extended to the newly proposed software-based Digital Subscriber Line (soft DSL) [22]. While CPU requirements for soft DSL are much higher, they possess some of the same real-time characteristics as soft modems, making them ideal candidates for understanding the benefits and limitations of real-time schedulers.

8. INDUSTRY PERSPECTIVES AND THINGS TO COME

8.1 Industry Perspectives on Implementation Choices

While one might assume that the vendor was unaware of or chose to ignore the *PC 99* timing guidelines [11] for soft modem interrupt handlers, we have learned through private communication [Anonymous soft modem vendor, September 2000—identity withheld due to licensing terms] that the real story is more complex (and more interesting). The vendor, in fact, did produce a version of their driver that performed signal processing in a thread, and tested this version with numerous combinations of hardware and other software.

Just like our THR implementation, their thread-based implementation normally worked fine. However, during testing, they came across a few scenarios that starved the modem thread. These included copying data from one IDE device to another (for instance, a CD to a disk), and starting applications such as Internet Explorer. They also saw a USB scanner using the Intel 440BX chipset holding off interrupts for 30-50ms. Therefore, in an understandable move of self-defense, they chose to do signal processing in interrupt context.

One might ask how such choices could be avoided, particularly since all users of this soft modem suffer the consequences of the signal processing being done in interrupt context, whereas only a very small number of test cases produced problems for a thread-based version. And indeed, those test cases were caused by behaviors themselves not conforming to the *PC 99* guidelines—behaviors such as IDE disk drivers not using DMA, for instance.

The vendor acknowledged that they would be thrilled to be good system citizens and run the signal processing in a thread, provided they could have confidence that other software and hardware vendors would do the same. Clearly, all would benefit from such an outcome, as overall system predictability would improve, including for the modem device itself.

To accomplish this, it is our belief that, ultimately, systematic latency timing verification of all components and software is the only viable solution. The definition of "correct operation" must be extended to include not just "produces the correct answer" but also measures of timeliness. Only then will vendors have the confidence to "play by the rules" because they know that others are as well. For more on this topic, see [6] and [15].

8.2 Soft DSL and Other Soft Devices

Software-based Digital Subscriber Line (soft DSL) [22] interfaces are about to appear on the market. The CPU requirements for soft DSL will be even more demanding than for soft modems. The facts below were obtained through private communication with the soft modem vendor [Anonymous soft modem vendor, September 2000—identity withheld due to licensing terms].

There are two communication rates for DSL: *G.lite*, which is 1.531Mbps downstream and 512Kbps upstream, and full rate DSL, which is 3.062Mbps downstream and 512Kbps upstream. Soft G.lite produces a CPU load of approximately 25% of a 600 MHz Pentium III. Full-rate DSL requires nearly twice that. For both rates, soft DSL requires a 4ms processing period—significantly shorter than the 12.5ms steady state period required for the V.90 soft modem.

Soft implementations of the 802.11b wireless LAN protocol [10] and the Bluetooth wireless protocol [9] are also possible. While only 2-3% of a 600 MHz CPU is needed, they require short computations extremely frequently—every 312.5µs.

Going further, software radio [4], including the use of adaptive modulation techniques only possible with soft implementations, is an active area of research and development activity.

The requirements of these, and other soft devices that may be developed, only increase the motivation for effective operating system and testing support for low-latency predictable computations.

9. CONCLUSIONS

We created four different versions of a soft modem driver that execute the signal processing code in interrupt context, in a DPC, in a thread using the Windows 2000 scheduler, and in a thread scheduled by the Rialto/NT real-time scheduler. We analyzed the performance profiles and the behavior of each driver version. Based on this analysis, we drew the following conclusions.

First, signal processing in interrupt context is not only unnecessary, but also detrimental to the predictability of any coexisting activity. Unfortunately, this is precisely what the vendor version does. We believe, however, that the vendor's choice is understandable given the absence of predictability guarantees in Windows 2000.

Second, the DPC version has some of the same predictability drawbacks as the vendor version. Both the vendor and the DPC versions do not conform to the *PC 99* set of recommendations for the Windows 2000 driver writers [11].

Third, the Windows 2000 scheduled thread version alleviates some of these problems. We found that the soft modem driver

functions well when the signal processing thread has high real-time priority and no competition.

Fourth, we conclude that other threads are less interfered with when the modem is scheduled using the real-time CPU Reservations abstraction. In particular, this abstraction allows us to control the amounts of time that the modem interferes with other time-sensitive computations while still meeting its needs.

In summary, this study makes the detailed performance characteristics of a popular soft modem available to the industry. We believe that this data should prove useful for informing ongoing work on providing predictable execution on consumer and general-purpose operating systems.

ACKNOWLEDGMENTS

We wish to thank the soft modem driver engineers for providing the driver source code for these experiments. Their assistance in clearing up some of the confusing modem behavior was invaluable. We also thank John Douceur, Steve Gribble, Patricia Jones, John Regehr, and the SIGMETRICS reviewers for their helpful comments on earlier drafts of the paper.

REFERENCES

[1] *3Com V.90 Technology*. 3Com Corporation, 1998. http://www.3com.com/technology/tech_net/white_papers/pdf/50065901.pdf.

[2] *Enterprise Remote Access Products*. 3Com Corporation, 2000. http://www.3com.com/products/remote.html.

[3] Douglas Anderson, Patrick Dawson, and Michael Tribble. *The Modem Technical Guide, First Edition*. Micro House International, June 1996.

[4] Vanu Bose, Mike Ismert, Matt Welborn, and John Guttag. Virtual Radios. In *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, pages 591-602, April 1999.

[5] Bill Carpenter, Mark Roman, Nick Vasilatos, and Myron Zimmerman. The RTX Real-Time Subsystem for Windows NT. In *Proceedings of the USENIX Windows NT Workshop*, Seattle, WA, pages 33-37, August 1997.

[6] Erik Cota-Robles and James P. Held. A Comparison of Windows Driver Model Latency Performance on Windows NT and Windows 98. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation* (OSDI '99), New Orleans, LA, pages 159-172, February 1999.

[7] *Software Modems and Microsoft Windows 2000*. Dell Corporation, December 1999. http://www.dell.com/us/en/hied/topics/vectors_1999-softmodems.htm.

[8] *DataFire RAS, Scalable Server-Based Remote Access Concentrators for Analog and Digital Connections*. Digi International, 2000. http://www.digi.com/solutions/mmcommadapters/dfras.shtml.

[9] Jaap Haartsen and Sven Mattisson. BLUETOOTH: A New Radio Interface for Ubiquitous Connectivity. In *Proceedings of the IEEE*, October 2000.

[10] IEEE Std. 802-11.1997, *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*. Approved 26 June 1997.

[11] *PC 99 System Design Guide – A Technical Reference for Designing PCs and Peripherals for the Microsoft Windows Family of Operating Systems, Chapter 19 – Modems*. Intel Corporation and Microsoft Corporation, 1998. ftp://download.intel.com/design/pc98/pc99/Pc_99_1.pdf.

[12] *Recommendation V.90 – A digital modem and analogue modem pair for use on the Public Switched Telephone Network (PSTN) at data signalling rates of up to 56000 bit/s downstream and 33600 bit/s upstream*. International Telecommunication Union, September 1998. http://www.itu.int/itudoc/itu-t/rec/v/v90.html.

[13] Michael B. Jones, Joseph S. Barrera III, Alessandro Forin, Paul J. Leach, Daniela Roşu, Marcel-Cătălin Roşu. An Overview of the Rialto Real-Time Architecture. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, Connemara, Ireland, pages 249-256, September 1996.

[14] Michael B. Jones, Daniela Roşu, Marcel-Cătălin Roşu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, St-Malo, France, pages 198-211, October 1997.

[15] Michael B. Jones and John Regehr. The Problems You're Having May Not Be the Problems You Think You're Having: Results from a Latency Study of Windows NT. In *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems* (HotOS-VII), Rio Rico, Arizona, IEEE Computer Society, March 1999.

[16] Michael B. Jones and John Regehr. CPU Reservations and Time Constraints: Implementation Experience on Windows NT. In *Proceedings Third USENIX Windows NT Symposium*, Seattle, WA, pages 93-102, July 1999.

[17] Michael B. Jones and John Regehr. *Predictable Scheduling for Digital Audio*. Microsoft Research Technical Report MSR-TR-2000-87, December 2000.

[18] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. In *Journal of the ACM*, vol. 20, pages 46-61, January 1973.

[19] *WDM for Windows 98 and Windows 2000*. Microsoft Corporation, 1998. http://www.microsoft.com/hwdev/desinit/WDMview.htm.

[20] Jason Nieh, James G. Hanko, J. Duane Northcutt, and Gerald Wall. SVR4 UNIX Scheduler Unacceptable for Multimedia Applications. In *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*. Lancaster, U.K., November 1993.

[21] David A. Solomon and Mark Russinovich. *Inside Microsoft Windows 2000, Third Edition*. Microsoft Press, 2000.

[22] Mike Tramontano. *The DSL Market is Going Soft*. Inter@ctive Week Online, Ziff Davis, July 17, 2000. http://www.zdnet.com/intweek/stories/news/0,4164,2604854,00.html.