IMPROVING REAL-TIME LATENCY PERFORMANCE ON COTS ARCHITECTURES

John Bono
Preston Hauck
NetAcquire Corporation

ABSTRACT

Telemetry systems designed to support the current needs of mission-critical applications often have stringent real-time requirements. These systems must guarantee a maximum worst-case processing and response time when incoming data is received. These real-time tolerances continue to tighten as data rates increase.

At the same time, end user requirements for COTS pricing efficiencies have forced many telemetry systems to now run on desktop operating systems like Windows or Unix. While these desktop operating systems offer advanced user interface capabilities, they cannot meet the real-time requirements of the many mission-critical telemetry applications. Furthermore, attempts to enhance desktop operating systems to support real-time constraints have met with only limited success.

This paper presents a telemetry system architecture that offers real-time guarantees while at the same time extensively leveraging inexpensive COTS hardware and software components. This is accomplished by partitioning the telemetry system onto two processors. The first processor is a NetAcquire subsystem running a real-time operating system (RTOS). The second processor runs a desktop operating system running the user interface. The two processors are connected together with a high-speed Ethernet IP internetwork. This architecture affords an improvement of two orders of magnitude over the real-time performance of a standalone desktop operating system.

KEYWORDS

Telemetry, COTS, Real Time, Low Latency, Deterministic, Distributed Systems

INTRODUCTION

Commercial-Off-The-Shelf (COTS) systems are attractively priced due to their volume production that lowers per-unit costs. This is a true cost savings when the user application of a unit perfectly meets what was the vendor's intended application when the system was produced. For example, a new desktop PC with a Microsoft Windows XP Home Edition operating system (O/S) is currently available for under $500. This is an incredible bargain for a home use PC.

For an office application, a faster CPU, more RAM, and the Windows XP Professional O/S are needed, adding $1000 to the cost. Thus, tailoring the computer to the application has tripled the cost, but this is still a good buy. For use in a telemetry application, such as processing data in a Telemetry Data Center, the plastic case needs to be replaced with a metal rack mounted chassis that includes extra fans and air filters as designed for this industrial application. The cost of the system rises incrementally again for this higher-reliability form factor. Windows runs on this industrial PC, so Windows can be kept, right? Well, actually it shouldn't.

Microsoft Windows is a wonderful operating system for desktop and server use. It provides an environment that has a great graphical user interface (GUI), the best software development tools available, and state-of-art networking support for just about any networking protocol. Various Unix and Linux systems are equally capable of running on an industrial PC and have nearly the same level of capabilities for the GUI, development tools, and networking support.

If these operating systems are so powerful, why can't they be used exclusively for a telemetry application? The answer is that a desktop operating system has these limitations when used in mission-critical telemetry applications:

- Lack of real-time performance
- Inherent instability due to sheer size of code base and mixed heritage third-party software components
- Inappropriate interactive nature through reliance on user prompts
- Intolerance to power loss and other hardware faults
- Vulnerability to hacker attacks due to widespread availability
- Difficulty of remote management without a connected keyboard and monitor

This paper focuses solely on the first issue, lack of real-time performance: the ability to read in multiple telemetry data streams, perform data processing, and output results in a guaranteed, time-bound fashion. "Latency" refers to the delay between receipt of input data and the creation of final output results – desktop operating systems lack the low latency characteristics required by many telemetry applications.

MISSION CRITICAL REAL-TIME APPLICATION EXAMPLES

Presented here are two applications that have fixed, low latency requirements: a radar range and communications with a low-earth-orbit satellite.

A radar range is a geographically large area containing multiple radar and telemetry antennas and ground stations. The range is used to test new and improved vehicles such as aircraft and missiles. The vehicles have on-board acquisition of flight parameters and other data that are combined into telemetry streams and sent out via RF transmissions. Multiple ground stations will monitor the vehicle's telemetry stream.

Simultaneously, multiple radar systems will track the vehicle's position, velocity and direction. In the case of an autonomous missile, the position data in the telemetry stream (where the missile thinks it is) and the radar's tracking position of the vehicle (where the missile really is) is compared in real-time. For safety reasons, a discrepancy outside of a small band between the two positions is grounds for an immediate abort of the test flight and destruction of the missile.

Given the speed of missiles, it is critical for the flight test operator to be given the position information as fast as it is acquired. In other words, the lowest possible latency of the telemetry processing system is essential. A "latency budget" is allocated out amongst the various components of the system, and the telemetry processor typically gets only a small portion of the overall budget. In one deployment of the NetAcquire product, the absolute worst latency was required to be under 10 milliseconds on all of 32 simultaneously processed data streams, and a typical latency of 1 millisecond was desired.

Depending on the mission requirements of the range test, there also may be a requirement to process data from multiple streams into an aggregate value. In these cases, it is critical that time-correlation be maintained across the channels. Small discrepancies in time can produce invalid results. Some ranges will also have a requirement to automatically detect the "best" source of the data, switching between multiple antennas that are all receiving the same data. The NetAcquire Correlating Source Selector (CSS) performs this function by comparing the various streams' data to each other. The data must be read quickly, the comparison done, and the best data must be output, all with as little latency addition as possible.

Another example of low latency requirements is a low-earth-orbit (LEO) satellite. A LEO satellite will come into view of a given ground station for only about ten minutes on each orbital pass. The very first thing that occurs during a pass is for the telemetry data to be analyzed to ensure that the satellite position and orientation are correct and if not, corrections are commanded. Then the high volume payload data starts streaming in. It is usually the case that the same unit handles both the telemetry data and the payload data. The latency of the telemetry data processing must remain low even though there is now a high volume of payload traffic handled by the same processor. In fact, in many applications even the payload traffic must be processed with low latency. With the current generation of communications and multimedia handling spacecraft systems, there is much more data than ever before and it travels through complex multi-hop networks, but users still demand low end-to-end latency that avoids noticeable communications delays during interactive voice and video communications.

In the case of both of these example applications, the addition of conventional telemetry system functionality can further negatively impact latency. As an example, there is often a requirement to store all the raw telemetry data for later off-line processing. For efficiency reasons, this often needs to be done on the unit receiving the incoming telemetry stream, and the data is written to a local hard disk. This must occur in parallel with the primary data processing, but without special operating system support, disk accesses will impact the latency performance of the primary real-time processing.

An additional challenge is that signal processing advances have allowed steady increases in satellite data rates, which in turn has an impact on telemetry processing systems. Individual data items come in faster and just as important, higher-fidelity data are acquired. This implies that, as the data rate increases, the user expectation of latency decreases. Some newer missile intercept telemetry systems are now using 500 Hz frame rates and requiring response times that are within one frame period, 2 ms!

OPERATING SYSTEM ADD-ONS

Typically there is a large variation between "average" latency and "worst case" latency; this variation can be as large as a factor of one hundred times. Windows or Unix may seem to do okay "a lot of the time" but the true worst-case latency is much worse and is simply unacceptable. Software vendors have recognized that desktop operating systems do not meet the needs of many telemetry and data acquisition applications. The desire to leverage the richness of the desktop O/S is strong and so the natural response is to see if it can be adapted to become more embedded and more real-time. Several add-on software products are available that provide near real-time capabilities to Windows or Linux. These work acceptably in certain industrial applications that do not have stringent real-time requirements or complex processing.

A major issue with operating system add-ons is that they do not address the fundamental problem that a desktop operating system kernel is too complex to be turned into a preemptible real-time operating system kernel. Portions of a desktop operating system can be made more responsive, but major portions of the operating system like networking, disk I/O, and alarm processing cause the system to drop out of real-time. Since operating system functions like networking and data archiving are required in most modern-day telemetry systems, an operating system add-on doesn’t provide the hoped-for real-time gains.

Another problem with these add-ons and their variants is that the resulting system loses the ease of development that the original desktop systems had. A part of the application works just like it would on Windows but part of it doesn’t. Which is which isn’t usually obvious. When the application is finished, what real-time performance will it really have? Furthermore, due to the complexity of a desktop operating system and the number of background tasks in the system, it is quite difficult to empirically determine the true worst-case latency by simple testing. Complex and extended-duration stress tests are needed to try to observe which is hopefully the true worst-case performance.

A LOW-LATENCY LAYERED ARCHITECTURE

An architecture for a telemetry system can be described as a set of hardware and software “layers” built upon each other. The descriptions below are provided roughly in a bottom-up order.

Base Hardware Layer

The best thing about the Windows operating systems is that they have caused PC manufactures to standardize the hardware-to-software interface. Thus, all PCs look similar to the low level software in regards to such things as CPU instruction set, BIOS memory locations, peripheral buses, etc. This has led to high volume production, lots of competitive hardware production, and thus low hardware costs. It is this standardization of PC hardware that allows other operating systems, such as Linux, to run on inexpensive computer platforms. This desktop hardware-interface standardization has even been driven into the industrial PC market and that is why Windows and Linux will run on many industrial computer platforms. In turn, the industrial PC vendors have brought their costs down by using high-volume desktop CPU chips, bus bridge chips, I/O chips, connectors, etc. Driven by the desktop market’s insatiable desire for faster and faster PCs, the hardware in these industrial computers has state-of-art capabilities. Due to the low cost, high performance and standard peripheral buses, an industrial computer is the perfect hardware platform for the foundation of a robust telemetry-processing architecture.

Operating System Layer

Not just desktop operating systems, but also true real-time operating systems (RTOS) can leverage the fact that the hardware-software interface of industrial computers is standardized. An RTOS vendor can produce their software with the ability to run on many different hardware vendors' computers. This increases the RTOS sales, and leads to higher volumes and lower per-system costs.

A true real-time operating system is designed from the very start with the primary goals of having the highest possible performance in areas such as:

- Interrupt Service Routine (ISR) response time
- Pre-emptive, priority-based multi-threading at the kernel layer
- Fast task switching time
- Priority-aware mutex, semaphore, and other inter-process communication/scheduling mechanisms
- Use of non-virtual memory for predictable and timely memory management
- Small thread time slices

The use of small thread time slices is of special note. Windows typically services each process at about 200 Hz (every 5 milliseconds). This clearly cannot meet the 2 ms latency requirement of the missile intercept application above. In contrast an RTOS, such as is used on a NetAcquire system, can time slice at up to 50,000 Hz (every 20 ***micro***seconds) giving it one-hundred time slices per data frame to do the required internal processing. In fact, the time slice size is something that is tune-able in a RTOS so that the latency can be traded off against a little more overall total throughput.

With its primary focus on performance, an RTOS is a far better choice than a GUI-centric desktop O/S when designing a telemetry front-end system. Telemetry trends are toward faster telemetry rates and higher densities of channels per unit. Desktop O/S trends are toward more GUI features and complex transaction-processing subsystems like Microsoft .NET. These trends mean that the overall latency difference between an RTOS and a desktop O/S will continue to grow. With each generation, desktop O/S's continue to improve, but never in the direction of providing "good enough" operation in low latency telemetry applications.

Another advantage of an RTOS is that it is designed for embedded, 7x24 "always on" applications. Unlike a desktop operating system with millions of lines of code, an RTOS consists of a carefully controlled execution environment designed for mission critical applications. It is small, exhaustively qualified, and purpose-built for mission-critical front-end processor activities.

An RTOS is not perfect however, and its intentional focus on real-time responsiveness usually means that less attention is paid to areas such as support for a wide variety of peripheral hardware drivers and for GUI user interfaces. Addressing these limitations is discussed in the following sections.

Driver/Peripheral Layer

Just as the Base Hardware layer benefits from production volumes, so can many of the peripheral hardware interfaces. Off-the-shelf cards for Ethernet, IEEE 1394 Firewire, and even PCM serial I/O are commonly available. Less known but still common are mezzanine approaches to I/O such as Industry Pack (IP) and PMC where one or more daughter cards resides on a larger carrier card that adapts it to the main computer bus.

Most of the time, needing a driver for a RTOS means writing it from scratch. In order to preserve COTS cost efficiencies, it is important to have a mechanism for developing new drivers for telemetry hardware in a timely fashion.

Telemetry hardware vendors will almost certainly have either a Windows or a Unix drivers available. However, Off-the-shelf Windows/Unix drivers from hardware vendors rarely meet the low latency requirements since the vendor’s primary focus is on the hardware production, not the driver. In addition, desktop operating system drivers are normally optimized for maximum throughput without concern for low latency.

NetAcquire Corporation has a long history of producing real-time hardware drivers, and the NetAcquire real-time operating system has an abstracted real-time object-oriented driver architecture that makes the task of creating real-time drivers much more direct. For example, a capability of this abstraction is to automatically break up large data transfers into small ones in order to avoid “hogging” the peripheral bus and to preserve low-latency across many incoming data channels. This streamlined device driver development architecture means that a wide spectrum of low-latency drivers is available for the NetAcquire RTOS. Interfaces such as analog/digital I/O, serial and PCM I/O, MIL-STD-1553, IEEE 1394 (Firewire), Gigabit Ethernet and many others are available as COTS offerings.

Data buffering is key aspect of real-time device drivers. For example, a PCM serial input card may synchronize to the start of a frame and then convert the serial stream into multiple-bit bytes or words read by the CPU. At slow data rates, the frame sync and serial-to-parallel conversion can add quite a bit of latency. At high rates, the data may be presented with very little time between bytes read. Thus, the bytes/words are commonly put into a First In First Out (FIFO) hardware buffer so that the CPU can read multiple bytes per interrupt.

With this approach, the timing of the FIFO-data-present interrupt needs to be set with consideration of the operating system’s ability to service the interrupt. A shallow FIFO means low latency, but the O/S must service the interrupt frequently, taking the CPU away from other tasks. Even worse, with many serial channels, the O/S may not be able to quickly service multiple interrupts that occur at the same time and so the shallow FIFOs will overflow causing data loss. On the other hand, a deep FIFO results in longer times between interrupts, making the O/S job easier, but also then contributing significantly to latency. The best solution to this is to use an operating system that has very fast interrupt service response times, low task switching times, and high data transfer rates – in short a RTOS.

Network hardware device drivers deserve special mention. Operating systems support network peripherals like Ethernet with a number of software-implemented layered protocols called a network “stack”. The network stack that comes with a desktop O/S is not real-time responsive, in that it is not priority based and typically does large amounts of buffering. Anyone who has seen his PC cursor “go away” for awhile while the network activity light on the PC lights up has experienced this. A similar delay in a unit handling multiple PCM serial I/O streams means loss of incoming data and gaps in transmitted data streams. RTOS vendors are well aware of this problem and use network stacks that are carefully designed to not “go away” despite the existence of other active tasks, network retransmits, and variable latency at the computer interacting on the other end of the network.

Data Processing Layer

“Data Processing” is a term used to describe just about any kind of program running on a computer. This is because almost every application is unique in some way. So the primary consideration at this layer is flexibility. But with flexibility there is a danger of losing ease of use. In addition, there is a potential for inadvertently increasing processing latency through poor algorithm design. A good solution is needed to the flexibility/ease-of-use/latency issue.

Telemetry applications often can be specified in a special form that can be used to reduce latency, called “graphical data flow representation”. Graphical data flow representation allows an application developer to simply draw a diagram of the data processing flows within the telemetry application (i.e., connecting a “frame synchronization” icon to a “disk archiving” icon with a line and an arrow). This data flow representation provides the telemetry processor with high-level information about how incoming data should propagate through the system with minimum latency. A latency-aware implementation of a data flow diagram can ensure that buffering depth at each level is dynamically adjusted to reflect dynamic data flow demands during system operation.

Because most telemetry systems use networks to connect between various computers for display and data distribution purposes, and it is important to provide application-level network communications that are low latency. This is in addition to the previously discussed latency issues at the network device driver layer. At the application level, network output data streams that exit through the data flow processing sub-system should allow a user-selectable range of network communication options that offer tradeoffs between latency and other factors like reliability and cost. A common choice is whether to use Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) for transmitting IP-based data. UDP has lower latency than TCP but is not as robust as TCP. On newer systems, a superset of both protocols can be used to achieve both low latency and high robustness, often called a “Publish/Subscribe” protocol. Publish/subscribe allows for individual data items to be “published” to the network as soon as they are available with no buffering delays associated with waiting for complete packets. Computers connected to the network can then “subscribe” to the published item by calling a simple software library function. Without any explicit network programming, the receiving computer’s local data variables are automatically kept up-to-date with the latest published data. The published data is transparently “pushed” from the publisher to the subscriber without any network polling operations and, if the publisher is running on an RTOS, with very low latency. In fact, a publish/subscribe protocol allows a low latency RTOS telemetry front-end to help higher-latency desktop O/S subscribers to achieve the best possible end-to-end latency. NetAcquire Corporation even offers publish/subscribe capability over specialized, extremely low latency communications interfaces like reflective memory (VMIC and SCRAMnet).

User Interface Layer

From the perspective of latency, the primary consideration of the user interface is that it not impede real-time processing. To do this, the user interface processing must be at the lowest priority level. This is exactly the opposite of how Windows or Unix works, where the O/S considers user interaction top priority.

To ensure that data processing is the highest priority, it is best to completely separate the user interface from the data processing. The user interface support should be handled in a separate task allowing it to run at a lower priority than the data processing. In Windows NT 3.5, Microsoft had the user interface separated this way. However, they found that for desktop applications, the responsiveness of the user interface wasn't good enough, so in Windows NT 4.0 they mixed the user interface back into the core O/S. This is the opposite of what is needed for real-time data processing.

On the other hand, real-time operating systems have the disadvantage that they have less highly developed user interfaces than Windows or Unix. This is because real-time applications emphasize data processing over advanced user interfaces. Furthermore, over the years millions of desktop users of Windows and Unix have supported desktop operating system vendor investment in additional COTS user interface capability.

An architecture that achieves the best of both worlds is to run a telemetry application on two separate systems. The first system runs the time-critical telemetry acquisition, data processing, and the archive functions from a real-time operating system (RTOS). The second system runs a desktop operating system for its advanced user interface features. The two systems are connected together with a high-speed Ethernet IP Internetwork. If PC hardware is used for both systems, low-cost COTS pricing is maintained across the entire system, while still supporting the partitioning of each task onto the operating system best suited to process it.

One powerful way of achieving a transparent network separation between a system optimized for real-time operations and a system providing a user interface is through the use of Web-based communication. In this architecture, the real-time system runs an embedded Web server task for providing data to one or more user interface systems running standard desktop Web browsers. The real-time operating system can carefully control the priority of the Web server to ensure that it does not interfere with real-time data processing. On the user interface side, the application achieves all the advantages of a graphical/mouse ease of use of a modern PC.

The desktop operating system uses a Web browser to display graphical user interfaces in both HTML and Java – since these technologies run on any operating system, there is not a restriction on the type of machine that can be used for user interfaces. Also, since Web architectures are inherently multi-user, more than one system serving as a user interface can simultaneously receive data from a single real-time system. CPU-intensive user interface operations like graphical rendering and interactive updates can thus run on one or more dedicated user interfaces, and this further frees the real-time time processor to dedicate all of its CPU cycles to hardware interface tasks and real-time data processing. Of course, a network-based architecture also removes any distance limitation: telemetry front-end processing can be separated from the user interfaces by several feet or by thousands of miles.

In a distributed multi-processor architecture, it is important to achieve very efficient network communication between the real-time system and the user-interface system(s). This requires multiple communications protocols between the two systems, not just HTTP. The publish/subscribe protocol described earlier is one such protocol. In addition, NetAcquire systems use a system-wide CORBA-based distributed "registry" of configuration and control parameters. This registry allows very efficient configuration and control of a large network of eal-time telemetry front-end processors.

Summary of Low Latency Telemetry System Architecture

Below is a summary of the architecture described above – it leverages many of the technologies of the commercial PC market, abandons those that compromise robustness, and costs only a small fraction of the $100K cost of non-COTS telemetry systems used in the past.

Layer	**Architecture**	**Benefit**
User Interface	Partition application into a real-time component on one system and a user interface component on another system. Remote access using a Web browser. Efficient distributed protocols for data distribution and configuration and control.	Combination of advanced user interface with guaranteed real-time performance. Multiple user interface locations, no distance limitations.
Data Processing	Graphical data flow architecture for configuring front-end processing. Latency-aware implementation that dynamically optimizes data propagation from input to output.	Low-latency processing while preserving convenient graphical configuration of applications without programming.
Peripheral/Driver	Leverages COTS PCI peripheral devices. • Low latency, abstracted device drivers developed for RTOS. Programmable FIFO depth vs. ISR frequency for latency optimization. Network support of different hardware: 10/100BaseT Ethernet, Gigabit Ethernet, Reflective Memory. Network support for different software protocols: TCP, UDP, publish/subscribe HTTP, SCPS, CORBA protocols Special network protocol stack designed for hard real-time operations.	Lowest latency without risk of CPU overload. Faster adaptation to new interfaces. COTS pricing. Many choices for communication approach.
O/S	Real-time operating system for I/O and processing in conjunction with desktop operating system for user interface.	Lowest latency, higher data rates, higher density channels, COTS pricing.
Base Hardware	Industrial ruggedized computer based on PC hardware-software interfaces.	COTS pricing. Multiple hardware suppliers.

CONCLUSION

This paper has shown examples of real-time telemetry applications which require low latency as well as evidence that desktop operating systems are not able to address these requirements, even with add-on “soft real-time” extensions. We have also shown that COTS pricing efficiencies are achievable by partitioning the system into a real-time telemetry processor and a desktop computer handling only the user interface. The NetAcquire telemetry processor leverages COTS hardware, a true real-time operating system, real-time aware drivers and network stack. These qualities combine to provide the lowest possible cost system that is guaranteed to meet very low latency requirements of current and future telemetry applications.