On Using Intelligent Network Interface Cards to support Multimedia Applications

Marc E. Fiuczynski† Richard P. Martin‡ Tsutomu Owa* Brian N. Bershad†

† Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195

‡ Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720

* Toshiba, Japan. Visiting Researcher at Dept. of Comp. Sci. and Eng., University of Washington, Seattle, WA 98195

**Abstract:** The emergence of fast, cheap embedded processors presents the opportunity for inexpensive processing to occur on the network interface. We are investigating how a system design incorporating such an *intelligent* network interface can be used to support streaming multimedia applications. We are developing an extensible execution environment, called SPINE, that enables applications to compute directly on the network interface and communicate with other applications executing on the host CPU, peer devices, and remote nodes. Using SPINE, we have implemented a video client that executes on the network interface, and transfers video data arriving from the network directly to the region of frame buffer memory representing the applications window. As a result of this system structure the video data is transferred only once over the I/O bus and places no load on the host CPU to display video at aggregate rates exceeding 80 Mbps.

# 1 Introduction

Multimedia applications often move large amounts of data between devices (e.g., between network and screen, disk and network, or disk and screen), and therefore place high I/O demands on both the host operating system and the underlying I/O subsystem. We observe that the exponential growth of processor speed relative to the rest of the I/O system presents the opportunity for application-specific processing to occur directly on intelligent I/O devices. Several network interface cards, such as the Myrinet, Alteon's Gigabit Ethernet ACEnic, and $I_2O$ systems, provide the infrastructure to compute on the device itself. With the technology trend of cheap, fast embedded processors (e.g., StrongARM, PowerPC, MIPS) used by intelligent network interface cards, the challenge is not so much in the hardware design as in a redesign of the software architecture needed to match the capabilities of the raw hardware. We intend to apply technology from the SPIN extensible operating system (use of a safe language, extensible interfaces, and resource management) to the network interface, to allow applications to customize their interaction with the network. The motivation is to move application-specific functionality directly to the network interface, thereby reducing I/O related data and control transfers to the host system.

Multimedia applications we believe will benefit from this architecture range from high-definition video clients, multimedia editors, Video-on-Demand servers (e.g., [6]), and 3D distributed real-time rendering (e.g., [7]). Other applications that may benefit, though not directly related to multimedia, range from packet filtering (e.g., Lazy Receive Processing [2]), cluster based storage management (e.g., Petal [3]), to distributed memory management systems (e.g., Global Memory [4]).

Our strategy is to provide an extensible runtime environment, called SPINE [9], for programmable network interface cards. Extensibility is important, as we cannot predict the types of applications that may want to process directly on the network interface. SPINE extends the fundamental ideas in SPIN [1] -- type safe code (called extensions) downloaded into a trusted execution environment -- to the network interface. Specifically, SPINE has three properties that are key to the construction of application-specific solutions:

- **Runtime adaptation and Extensibility.** An application, regardless of its privilege level, may define a SPINE extension and dynamically load it onto an intelligent I/O card.
- **Performance.** SPINE extensions run in the same address space as the firmware, with low-overhead access to services and hardware, and don't require the host CPU to be interrupted. Extensions may share data using peer-to-peer DMA and communicate via message queues. Finally, overall system performance can be improved by reducing I/O related control and data transfers.
- **Safety and Fault Isolation.** The use of a SPINE extension does not compromise the safety of other extensions, the firmware, or the host operating system. Extensions are isolated from the system and one another through the use of a type-safe language, defensive interface design, and resource management.

The rest of this paper describes the software architecture of SPINE and the video application we have built using it.

# 2 SPINE Software Architecture

SPINE is a safe execution environment that enables applications, which require tight integration with the network, to specialize the network interface for its performance and functionality requirements. We do not assume that the processor architecture of programmable network interfaces provides support for multiple address spaces. A safe language enables extensions and SPINE to run in the context of a single address space. Our design provides safe interfaces to the underlying hardware and allows extensions to provide their own extensible services. In the next two subsections we describe the SPINE runtime and the SPINE communication layer in more detail.

## 2.1 SPINE Runtime

The SPINE runtime consists of a small Modula-3 runtime that provides support for threads, synchronization, and memory management. It is split across the network adapter and the host into I/O and kernel runtime components, respectively. The basic system structure is shown in Figure 1. Applications define SPINE I/O extensions in Modula-3, which is a Pascal-like type safe programming language. A

SPINE extension may be dynamically loaded onto the network interface card (see the arrow labeled 1 in Figure 1.) where they link with the SPINE I/O runtime. We envision that for some applications it will be necessary to tightly couple both kernel and I/O extensions, and our goal is to support this scenario using a similar extension mechanism. Thus, optionally, applications may define SPINE kernel extension, also in Modula-3, that are loaded into the kernel's virtual address space (see arrow labeled 2 in Figure 1.) where they link with the SPINE kernel runtime.

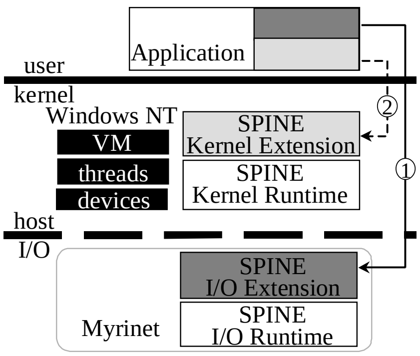


**Figure 1.** SPINE System Structure.

The SPINE *kernel runtime* provides host-side services to the SPINE *I/O runtime*, such as initializing the intelligent NIC and loading extensions onto the card. Additionally, it provides access to the operating systems thread, device and virtual memory subsystems for SPINE kernel extensions.

The *SPINE I/O runtim*e, which is also implemented in Modula-3, is the execution environment that enables application to compute on the NIC. It exports internal and external interfaces. Internal interfaces are used by extensions that are loaded onto the network interface, which consists of the standard Modula-3 interface, "plain old kernel services", and safe access to the underlying hardware (such as access to DMA engines). The external interface consists of a message FIFO that enables user-level applications, peer devices, and kernel modules to communicate with extensions on the network interface using an active message communication layer. The SPINE I/O runtime manages resources on the network interface. For example, the buffers used as the source/sink of DMA operations are one such resource. These buffers are accessed using unforgable references (i.e., an extension cannot create an arbitrary memory address as a DMA source or destination address), which may refer to local-, peer device-, or host-memory.

## 2.2 SPINE Communication Layer

The SPINE communication layer is a variation of Active Messages [5] used by the NOW project at UC-Berkeley. It differs from other active message layer implementations in that it can execute shared-memory operations in the context of the network interface. This message layer has the flexibility to execute extension code either on the network interface or the host CPU. The SPINE I/O runtime implements an active message dispatcher that determines whether the message should be pushed to the host or to invoke the handler of a local SPINE extension. SPINE extensions register active message handlers with the I/O runtime, which are invoked when a message arrives from the network, the host, or possibly an intelligent peer device. All handlers are invoked with exactly two arguments: a context variable that contains information associated with the handler at installation time; and, a pointer to the message that contains the data as well as control information specifying the source and type of the message. There are two types of messages: small messages and bulk messages. Small messages are currently 64 bytes total. Bulk messages are similar to small messages, but contain a reference to a data buffer managed by the SPINE I/O runtime. Finally, a handler associated with the active message is invoked after all of the data for the message has arrived.

# 3 Video Client Application

We use Windows NT 4.0 running on a Pentium based PC as the host system. As the intelligent network interface we use Myricom's Myrinet card on the PCI bus, containing 256 KB SRAM card memory, a 33 MHz "LANai" processor, with a network wire rate of 160MB/s. The LANai processor is a general-purpose processor that can be programmed with specialized control programs and plays a key role in allowing us to experiment with moving application-specific functionality onto the network interface. We have developed our own firmware for the Myrinet upon which we layer the SPINE I/O runtime.

Using SPINE we have implemented a video client application that defines an application-specific *video extension*, which transfers video data arriving from the network directly to the frame buffer. The video client runs as a regular application on Windows NT. It is responsible for creating the framing window that will be used to display the video and informing the video extension of the window coordinates. The video extension on the network interface maintains window coordinate and size information, and DMA transfers video data arriving from the network to the region of frame buffer memory representing the application's window. The video client application catches window movement events and informs the video extension of the new window coordinates.

The implementation of the video extension running on the network interface is simple. It is roughly 250 lines of code, which consists of functions to a) instantiate per-window metadata for window coordinates, size, etc., b) update the metadata after a window event occurs (e.g., window movement), and c) DMA transfer data to the frame buffer. These functions are registered as active message handlers with the SPINE I/O runtime, and are invoked when a message arrives either from the host or the network.

Figure 2 depicts the overall structure in more detail. The numbered arrows have the following meaning:

1. NetVideo application loads video extension onto card.
2. Packet containing video data arrives from the network.
3. SPINE I/O runtime dispatches packet to video extension.
4. Video extension transfers data directly to the frame buffer, and the video image appears on the user's screen.

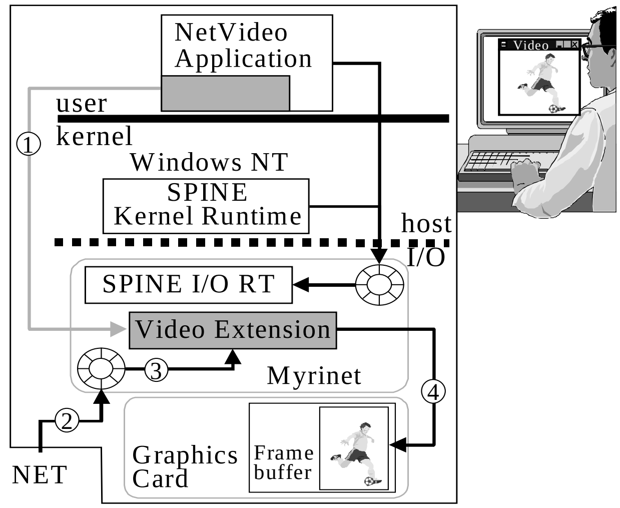


**Figure 2** Network video. The Video Extension transfers data from the network directly to the frame buffer.

Using this system structure the host processor is not used during the common case operation of displaying video to the screen. In our prototype system we've been able to support several video clients, each at sustained data rates of up to 40 Mbps, with a host CPU utilization of zero percent for the user-level video application. Thus, regardless of the operating systems I/O services and APIs, we can achieve high-performance video delivery.

The caveat in our current video client experiment is that, although the Myrinet's DMA engines can move large quantities of data, the LANai processor is to slow to decode video data on the fly. The LANai is roughly equivalent to a SPARC-1 processor, i.e., it represents roughly 1989 processor technology. Consequently, our video server takes on the brunt of the work and converts MPEG to raw bitmaps, which it then sends to the video client. Thus the video extension essentially acts as an application-specific data pump; taking data from the network and directly transfering it to the right location of the frame buffer. We expect fast, embedded processors to be built into future NICs that will enable on-the-fly video decoding, or one could use a graphics card that supports video decoding in hardware to avoid decoding on the NIC.

## 4 Summary

Commercial operating systems that we know, love, and use on a daily basis, may be slow to adopt new techniques to support digital audio/video. Intelligent NICs make it possible to implement application-specific functionality below the operating system, thereby enabling new techniques to be incorporated into commercial systems. A hardware design using a "front-end" I/O processor is not new, but traditionally has been relegated to special purpose machines (e.g., Auspex NFS server), mainframes (e.g., IBM 390 with channel controllers), or supercomputer designs (e.g., Cray Y-MP). The challenge is not so much in the hardware design as in a redesign of the software architecture needed to match the capabilities of the raw hardware. We are designing a flexible I/O execution environment, called SPINE, to enable applications to execute on the network interface. The video client is one example that leverages the SPINE software architecture to efficiently move data through the system. Other application-specific techniques, which may not be easily adopted by commercial operating systems (if at all), could be implemented on the NIC as well. For example, the video client could be improved with Jeffay's [10] buffer management techniques to reduce jitter in a congested network.

Intelligent network cards are particularly compelling when the processor used on an intelligent NIC tracks the relative performance of the host processor. We conjecture that link bandwidth of NICs used in conventional workstations will not exceed one gigabit over the next five years, while the performance of processors continues on its current growth path. Therefore, we expect that future intelligent NICs will be fast enough for a variety of application-specific extensions. For example, Alteon's ACEnic gigabit Ethernet interface has two 100MHz R4000 processors to allow for "… the opportunity to implement more intelligent, value-added features…" [8].

Finally, there are various additional issues to be considered when using this type of hardware design and software architecture. Here are some of the general questions that we intend to examine in the long term with our research:

- When does it pay off to move system functionality to the NIC? Is there enough capacity on the NIC to support interesting [multimedia] applications?
- How does one split applications across the host/IO boundary? Can the split be hidden by "middleware"? How do user-level applications and extensions synchronize?
- What level of OS support is required on the NIC? What types of real-time or QoS support do extensions require?

## References

[1] B.N. Bershad, S. Savage, P. Pardyak, E.G. Sirer, M.E. Fiuczynski, D. Becker, S. Eggers, C. Chambers. Extensibility, Safety and Performance in the SPIN Operating System. Proc. SOSP-15, Dec. 1995.

[2] P. Druschel and B.Gaurav. *Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems*. Proc. OSDI'96, Oct. 1996.

[3] E.K. Lee and C.A. Thekkath. *Petal: Distributed Virtual Disks*. Proc. ASPLOS-7, Oct. 1996.

[4] M.J. Feeley, W.E. Morgan, F. Pighin, A. Karlin, and H.M. Levy. *Implementing Global Memory Management in a Workstation Cluster*. Proc. SOSP-15. Dec. 1995.

[5] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. *Active Messages: A Mechanism for Integrated Communication and Computation*. Proc. ISCA-19, May 1992.

[6] W.J. Bolosky, J.S. Barrera III, R.P. Draves, R.P. Fitzgerald, G.A. Gibson, M.B. Jones, S.P. Levi, N.P. Myhrvold, R.F. Rashid. *The Tiger Video Fileserver*. Proc. NOSSDAV'96, Apr. 1996.

[7] Thu D. Nguyen and John Zahorjan. *Scheduling Policies to Support Distributed 3D Multimedia Applications*. To Appear Proc. SIGMETRICS '98, Jun. 1998.

[8] Alteon Networks. Gigabit Ethernet Technology Brief. Available from: http://www.alteon.com/techbr.html

[9] M.E. Fiuczynski. SPINE: A Safe Programmable and Integrated Network Environment. Overview Information available from: http://www.cs.washington.edu/homes/mef/research/spine

[10] K. Jeffay, D.L. Stone, F.D. Smith. Transport and Display Mechanisms for Multimedia Conferencing Across Packet-Switched Networks. Computer Networks and ISDN Systems. Jul. 1994.