# RiceNIC: A Reconfigurable Network Interface for Experimental Research and Education

Jeffrey Shafer and Scott Rixner
Rice University
Houston, TX
{shafer,rixner}@rice.edu

## ABSTRACT

The evaluation of new network server architectures is usually performed experimentally using either a simulator or a hardware prototype. Accurate simulation of the hardware-software interface within the network subsystem is challenging due to the interactions of multiple asynchronous systems. Small timing inaccuracies in such a system can perturb the hardware and software state yielding potentially misleading results. Hardware prototypes show more promise because they are real-world implementations, not simplifications. Existing Ethernet network interface cards (NICs) are unsuitable for prototyping as they lack the capability and/or flexibility for advanced networking research.

RiceNIC is an open network interface prototyping platform for public use. This reconfigurable and programmable Gigabit Ethernet NIC is designed to address the dilemma of how to accurately evaluate new ideas in network server architecture, and is built for use in experimental research and education. The flexibility and capability of RiceNIC has proven invaluable in recent research efforts.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; B.4.3 [**Hardware**]: Interconnections—*Interfaces*

## General Terms

Experimentation, Design

## Keywords

Network Interface Cards, Ethernet, FPGA Prototyping

## 1. INTRODUCTION

Networking has become an integral part of modern computer systems. While the network interface has traditionally been a simple device that forwards raw data between the network and the operating system, its role is changing. The field of network server architecture spans the network subsystem, from the operating system down to the local interconnect and network interface card (NIC). Of particular importance are the efficiency of the communication mechanisms between the operating system, including the device driver, and the NIC. As communication is arguably one of the most important aspects of modern computer systems, optimization of the network subsystem is particularly important. To improve overall networking performance, work in this field has produced sophisticated network interfaces that perform functions such as TCP offloading, iSCSI, encryption and firewalling, remote direct memory access, and many others. These innovations all span both hardware and software layers and include many complex interactions between the two.

Traditionally, such ideas are evaluated using either prototyping or simulation. Prototyping for research is frequently done using existing programmable network interfaces. These interfaces are typically underpowered for the proposed functionality, limiting the value of the prototype. The behavior of modern network protocols, such as TCP, is largely dependent on overall system performance and behavior. This means that it is difficult to accurately experimentally evaluate such underpowered prototypes, as the network protocols themselves will behave much differently than they will with a fully capable device.

Simulation is often used in experimental research to obviate the need for prototyping. However, simulation of the network subsystem requires accurate modeling of most parts of the host system architecture, and does not easily lend itself to a simplified model. The performance of the network interface depends on the behavior of the processor, memory system, local interconnect, and the design of the interface itself. All of these components operate asynchronously from each other, and thus accurate simulation requires complete models that correctly account for the complex interactions between these systems. In addition to the hardware, a complete software environment must be simulated, including the operating system and device driver. This is very important in network server architecture since many innovations involve making changes to both hardware and software architecture at the same time. Finally, the behavior of network protocols such as TCP are highly dependent on the behavior of both systems in communication as well as the behavior of the network. Even if the network server is accurately simulated, experimental evaluation still requires accurate simulation of the interaction between two TCP network stacks


*ExpCS* 13-14 June 2007, San Diego, CA

on separate systems connected by an inherently chaotic interconnect such as the Internet.

Given the challenges associated with network server architecture simulation, research using experimental hardware prototypes should be encouraged whenever possible. RiceNIC, a reconfigurable and programmable Gigabit Ethernet NIC, helps overcome these challenges. The NIC design is freely available and includes the FPGA bitstreams, firmware, and Linux device driver. It provides researchers with a flexible network interface that can be easily modified to support a wide range of research activities, providing a significant head-start compared with implementing an experimental prototype from scratch. It can also be used for experimental class projects.

The NIC includes multiple FPGAs, large on-NIC memories in excess of 256 MB, two 300 MHz embedded PowerPC processors, and a copper Gigabit Ethernet physical interface and connector. Using only a single PowerPC processor, RiceNIC is able to saturate the Gigabit Ethernet network link with maximum-sized packets. This leaves significant resources—including 50% of the reconfigurable logic elements on the Virtex-II Pro FPGA, a spare PowerPC processor, and hundreds of megabytes of memory—available on RiceNIC for use in experimental networking research.

RiceNIC was used in a network research project that shows both the value of having an experimental prototype and the value of using RiceNIC as a design platform. In this project, RiceNIC FPGA hardware and firmware was modified to allow multiple virtual machines running on the same system to concurrently control a single NIC, improving the efficiency of network virtualization. RiceNIC ran at full network speed, allowing highly accurate performance measurements to be taken that strongly demonstrate the improvements possible with the new virtualized network architecture. In addition, having a fully-capable hardware prototype allowed the experimental determination of the minimum data buffer size on the NIC to enable full network throughput even in a heavily loaded system. This helped prove that the proposed technique was feasible in even a low-cost NIC.

In the remainder of this paper, Section 2 introduces RiceNIC as an open prototyping platform for network server architecture research, and evaluates its performance. Then, Section 3 presents a case study on using RiceNIC for experimental research and Section 4 discusses how the NIC could be used in education. Section 5 discusses some of the challenges inherent in using simulators in network server architecture research, and presents prototyping as a valuable alternative limited by the currently available programmable NICs. Finally, Section 6 concludes the paper.

# 2. RICENIC PLATFORM

RiceNIC is a Gigabit Ethernet network interface card with open hardware and software specifications. It is specifically designed for experimental network server architecture research, and provides considerable design flexibility and numerous computational and memory resources. RiceNIC is built on the Avnet Virtex-II Pro Development Board shown in Figure 1. This FPGA prototyping board includes all of the components necessary for a Gigabit Ethernet network interface with embedded processors and on-board memory. The commercial availability of this board obviates the need to custom-design a similar board.

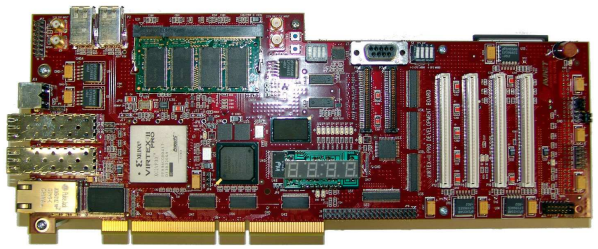

**Figure 1: RiceNIC PCI Card**

The architecture and interconnection of the FPGAs, memories, and other devices on the Avnet board is shown in Figure 2. The Xilinx Virtex-II Pro FPGA on RiceNIC contains most of the NIC logic, including the PowerPC processors, on-chip memories, MAC controller, DMA unit front-end, and DDR memory controller. The PHY and DDR memories are directly connected to the Virtex FPGA. Most components on the Virtex FPGA are interconnected by a processor local bus (PLB). The smaller Spartan-IIE FPGA contains the 64-bit, 66 MHz PCI controller, the back-end DMA controller, and a SRAM memory controller that is accessible by both the NIC and host system. Both the MAC and PCI units are built around low-level interfaces provided by Xilinx; however, those units are wrapped with a custom descriptor-based control system to integrate them into the rest of the system and to provide flexible software control over the hardware functionality. A complete description of the NIC design can be found in [21].

## 2.1 Facilities for Experimentation

RiceNIC has several design elements that make it particularly user-friendly for experimental applications, including a serial console to the embedded processors, timer-based profiler, software-controlled MAC and DMA hardware assist units, and large quantities of free memory and FPGA resources for new innovations.

*Serial Console.* Unlike any other commercially available network interface, RiceNIC provides a UART that is accessible over the PLB by the PowerPC processors. This UART interfaces with a serial port on the NIC that can be connected to an RS-232 serial port on an external computer, allowing terminal access directly to the network interface firmware. The firmware can display status and debugging information to the terminal and provide a command-line interface for querying the NIC as it operates. Such terminal access greatly facilitates debugging, development, and experimental evaluation of the network interface.

*Statistical Profiler.* A timer-based statistical profiler is included with the baseline firmware. This tool can be used to view the firmware execution time on a per-function basis, and thus determine the most frequently executed instructions. It operates similarly to statistical profilers for modern general-purpose processors. The profiler facilitates

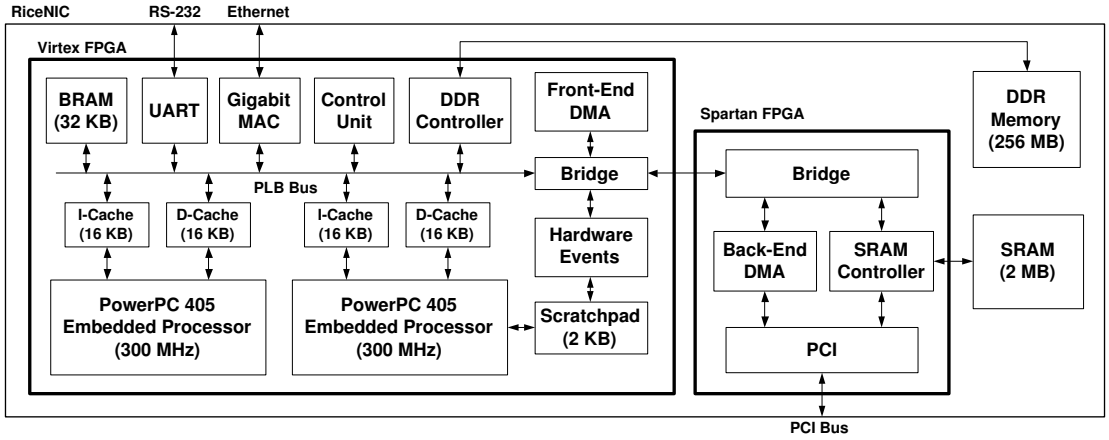


**Figure 2: RiceNIC System Architecture**

experimental evaluation and optimization of developmental network interface firmware.

*Media Access Control.* The media access control (MAC) unit is responsible for transferring data between NIC memories attached to the PLB and the physical interconnect (PHY) module. RiceNIC provides the PowerPC firmware significant flexibility in controlling the MAC operation through the use of descriptors. These 64-bit control descriptors are transferred to the MAC unit, stored in FIFO queues, and consumed by the MAC when it is able to process them. By using explicit descriptors under processor management, this architecture gives the firmware flexibility to manage the hardware functions of the MAC unit. Unlike conventional NICs, such as the Tigon 2 [1], that use a contiguous circular receive buffer, RiceNIC allows buffers to be placed arbitrarily in memory under firmware control. This allows the implementation and evaluation of advanced algorithms, such as out of order processing of received packets, that would be difficult if a simple circular buffer was mandated by the hardware.

An optional hardware unit can calculate the TCP or UDP checksum on both transmitted and received packets. For transmitted packets, the checksum is calculated when the packet is transferred from on-NIC memory to the MAC and placed directly in the outgoing datastream at a user-specified location. This allows the NIC processor to modify the packet (or create new packets entirely!) and still gain the performance benefits of a hardware checksum module.

The MAC unit also implements gather transmit and can assemble a single packet from discontiguous regions in NIC memory. This capability is very useful in NIC research such as TCP offloading where the NIC processor generates packets itself instead of merely forwarding data generated by the host system. In this scenario, the NIC does not need to copy packet data to a contiguous buffer for transmission.

*Direct Memory Access.* RiceNIC, like all modern NICs, transfers data to and from the host using direct memory access (DMA). A custom-built DMA assist unit transfers data in 2 KB bursts between NIC memory and main memory via the PCI bus. As with the MAC unit, the DMA unit is controlled by the firmware through the use of descriptors, allowing the firmware flexible control over the operation of the hardware, and enabling scatter/gather I/O in each direction. The DMA unit processes each descriptor and performs the data transfer asynchronously.

*NIC Memory.* RiceNIC includes substantial memory resources both in the FPGA fabric and on the NIC. Together these memories are orders of magnitude larger than those found on a conventional network interface, enabling storage-intensive NIC research. For general-purpose storage, 256MB of DDR-SDRAM is attached to the Virtex FPGA. 32 KB of on-FPGA BRAM and 16 KB processor data caches are provided for frequently accessed data. Finally, 2 MB of SRAM is attached to the Spartan FPGA.

The NIC memories have widely differing performance and capabilities. The SRAM has very high latency due to its remote location across the FPGA bridge, but is very useful as shared memory between the host system and the NIC because it is accessible via PIO from both. The DDR-SDRAM module provides the most storage capacity, but at a higher access latency than the small on-chip BRAM and processor cache. Given these differences, data can be placed in the appropriate memories based upon its size and the frequency with which it is accessed. Thus, RiceNIC gives the programmer substantial flexibility to explore design tradeoffs regarding NIC memory capacity, performance, and price.

*Available Resources.* The Virtex FPGA on RiceNIC still has substantial resources available for future research and development. The implementation of a fully functioning Gigabit Ethernet NIC consumes less than 40% of the embedded BRAM, less than 1% of the DDR-SDRAM, and less than 50% of the reconfigurable logic elements. This leaves significant design resources for the implementation of additional features.

The Spartan FPGA, however, is essentially filled to capacity in the current design. The NIC architecture shown in Figure 2 requires that the Spartan FPGA contain the PCI core and SRAM memory controller. Fortunately, it is likely

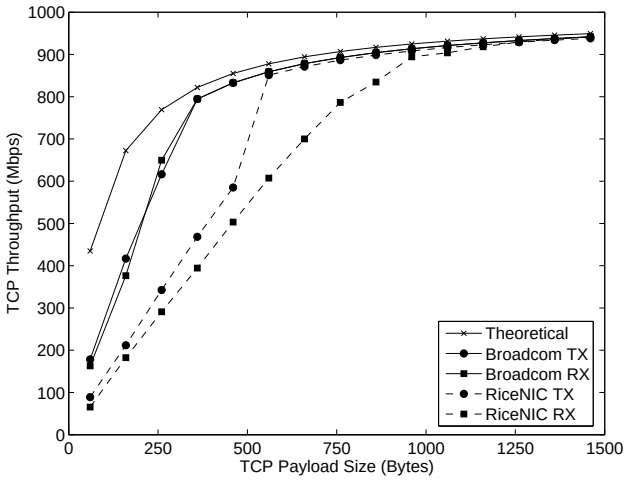


**Figure 3: Network Throughput**

that hardware enhancements would be most effective on the larger and faster Virtex chip. If additional logic resources were required on the Spartan, the DMA data buffers stored there could be reduced from their current 2 KB size at a cost of reducing the burst length and efficiency of PCI transfers.

## 2.2 Performance Evaluation

This section briefly discusses the performance of RiceNIC using a lightweight TCP streaming network benchmark between two similarly-configured Opteron servers running Linux 2.6. The server being tested used either the RiceNIC or a commercial Broadcom 5704C NIC, and the other endpoint used the same Broadcom NIC. The host systems were not the bottleneck in any test, and the TCP stacks were configured to saturate the NICs with low CPU utilization. Note that the MTU was specifically lowered to transmit small packets. Otherwise, Linux would merge multiple small TCP packets into one large packet before delivering it to the NIC for transmission.

RiceNIC was configured with firmware that implements a fully functioning Gigabit Ethernet network interface and only uses 1 of the 2 available PowerPC processors. For both NICs, checksum offloading was enabled.

Figure 3 shows the TCP throughput of RiceNIC, the Broadcom NIC, and the theoretical Ethernet limit for a range of TCP packet sizes. As the figure shows, RiceNIC closely tracks the performance of the Broadcom NIC, and the theoretical Ethernet limit, for packet sizes larger than 500 bytes when transmitting and larger than 900 bytes when receiving. For smaller packet sizes, RiceNIC is CPU limited by the higher packet rate, leading to lower throughput than the Broadcom NIC. Furthermore, RiceNIC's receive performance is less than its transmit performance because receive processing requires more per-packet computation to verify checksums. The second unused PowerPC processor on RiceNIC could be employed to increase throughput for smaller packet sizes. However, the network traffic of many applications is biased towards larger packets, so improving the throughput for smaller packets may not provide any application-level performance improvements.

# 3. RESEARCH WITH RICENIC

RiceNIC facilitates experimental evaluation of complex hardware and software changes to the network subsystem. This section introduces a research effort in which prototyping with RiceNIC was critical to the experimental process: concurrent direct network access for virtual machine monitors [25]. In addition, this section explores two recent research efforts — network interface data caching [12] and TCP connection handoff to the NIC [11] — that were limited by existing prototyping technology and would have benefited greatly from RiceNIC had it been available at the time. These case studies show the flexibility of RiceNIC for networking research and illustrate the value of a fully-capable experimental prototype.

## 3.1 Support for Virtualization

Virtual machine monitors (VMMs) allow multiple virtual machines running on the same physical machine to share hardware resources. To support networking, such VMMs must virtualize the machine's network interfaces by presenting each virtual machine with a software interface that is multiplexed onto the actual physical NIC. The overhead of this software-based network virtualization severely limits network performance.

Concurrent direct network access (CDNA) combines both software and hardware components to significantly reduce the overhead of network virtualization in VMMs [25]. The CDNA network virtualization architecture provides virtual machines running on a VMM safe direct access to the network interface. With CDNA, each virtual machine is allocated a unique *context* on the network interface and communicates directly with the network interface through that context. In this manner, the virtual machines operate as if each has access to its own dedicated network interface.

RiceNIC hardware and firmware were modified to enable a prototype implementation of CDNA. First, the SRAM controller was altered to divide the low 512 KB of SRAM memory into 128 independent *contexts* of 4 KB each. Each context is used by a separate virtual machine to transfer control data via PIO to the network interface, while bulk data is still transferred by DMA operations. A conventional network interface would only need a single context, but in order for the NIC to directly communicate with multiple guest operating systems in a virtual machine monitor, each guest needs its own control region. By making each context the same size as a physical page, 4 KB, the virtual machine monitor can map each context into the address space of a unique guest operating system. A custom hardware event notification system was also implemented on the FPGA to provide the firmware an efficient and low-latency method to determine when a context is updated by a virtual machine with new control information. This eliminates the need for the firmware to constantly poll each context searching for updated information.

In addition to modifying RiceNIC hardware, the NIC firmware was also modified to independently communicate with the guest operating systems through these contexts, as well as perform network traffic multiplexing and demultiplexing. This requires about 8 MB of additional NIC memory, which easily fits within the 256 MB DDR SODIMM.

Even with these modifications, it was still not necessary to use the second PowerPC processor. Finally, the VMM was modified to communicate with the firmware to provide memory protection among the guest operating systems and ensure that they do not direct RiceNIC to transfer data to or from physical memory that is not owned by that guest.

These modifications to RiceNIC firmware and FPGA hardware resulted in significant networking performance improvements for virtual machine monitors. Both the system throughput with a single guest OS, and the system scaling for increasing numbers of guest operating systems, were improved by significant margins [25]. These improvements would be difficult, if not impossible, to achieve with any other existing Ethernet network interface, and show the advantages of using RiceNIC for research into future network interface architectures.

The CDNA prototype on RiceNIC proved very useful in experimentally determining key architectural features, such as the minimum size of on-NIC packet buffers per virtual machine to fully saturate the Ethernet link. This was an important contribution of the published research, because it was found that the per-OS buffer requirements of 384 KB were well within the capability of a modern NIC and could be implemented at a low cost. Estimating the buffer size in a virtualized system is difficult, because buffer requirements are directly linked to non-obvious scheduling decisions by the virtual machine monitor that determine how frequently each guest operating system can execute. The less frequently an OS runs, the more buffering it needs to maintain network throughput. With the RiceNIC prototype, a series of focused experiments on a heavily-loaded virtualized system were conducted at at a wide range of buffer sizes in order to determine the best size for optimal performance. Performing these experiments in simulation would have been difficult because saturating the buffers required execution times in the order of tens of seconds, and most simulations are only run for brief timeslices in order to reduce simulation time.

## 3.2 Prior Projects

Several prior projects at Rice would also have benefited from the use of RiceNIC. In particular, resource constraints on existing network interfaces restricted work on network interface data caching [8, 12] and connection handoff for TCP offloading [10, 11]. These projects used a combination of Tigon 2 programmable network interfaces and full system simulation to evaluate innovations involving the network interface. The knowledge gained from these projects directly motivated the development of RiceNIC as a far more capable network interface than those based on the Tigon 2 architecture. RiceNIC would have enabled a complete prototype implementation of these innovations, allowing more sophisticated evaluations of these concepts in real systems.

### 3.2.1 Network Interface Data Caching

Network interface data caching reduces local interconnect traffic on network servers by caching frequently-requested content on a programmable network interface [8, 12]. The operating system on the host CPU determines which data to store in the cache and for which packets it should use data from the cache. To facilitate data reuse across multiple packets and connections, the cache only stores application-level response content (such as HTTP data), with application-level and networking headers generated by the host CPU. Therefore, packets are composed of multiple fragments, some of which reside in main memory and some of which reside in the memory on the NIC. The NIC must have enough processing power to properly assemble packets from these fragments. Furthermore, the NIC must have a large enough memory, on the order of 16 MB, to cache a sufficient amount of application-level data.

While the Tigon 2 had sufficient processing power to assemble packets on the NIC, it did not have enough storage to fully implement network interface data caching. In order to evaluate the idea, a larger memory was emulated by allocating a single page on the NIC as the cache. Despite the cache's small size, the operating system managed the cache as if it were much larger. Data was transferred to the cache and written into the single page based upon information held in the cache directory for a full-sized cache. All cached payload data was read from this single page, so outgoing packets did not contain the correct payload. While this emulation strategy provides accurate results—all data transfers are performed correctly—it yields incorrect traffic on the network. This made correctness and performance debugging difficult. RiceNIC would resolve this problem, as it has sufficient memory to evaluate cache sizes that are much larger than those considered in the original research.

### 3.2.2 Connection Handoff

Using connection handoff, the operating system can offload a subset of TCP connections in the system to the network interface, while the remaining connections are processed on the host CPU [10, 11, 15, 16]. Offloading can reduce computation and memory bandwidth requirements for packet processing on the host CPU. However, full TCP offloading may degrade system performance because finite processing and memory resources on the network interface limit the amount of packet processing and the number of connections. Using handoff, the operating system controls the number of offloaded connections in order to fully utilize the network interface without overloading it. Handoff is transparent to the application, and the operating system may choose to offload connections to the network interface or reclaim them from the interface at any time. As with network interface data caching, connection handoff requires increased processing and storage from the network interface. The NIC must perform TCP/IP processing and it must have sufficient storage to hold both data and state information for connections that have been handed off to the NIC.

The implementation of connection handoff requires modifications to the operating system and to the network interface. It also changes the communication interface between the operating system and the NIC. In order to guarantee that the operating system's behavior was functionally correct, connection handoff was first implemented on a prototype system. The initial prototype was developed with a network interface based upon the Tigon 2 architecture. As connection handoff was developed on a real system, it was possible to ensure that the operating system could correctly handle the complex asynchronous behavior of the modified network interface. Using a simulator would be far too slow to provide interactive feedback during the design process. Further-

more, it would be nearly impossible to guarantee that the modified operating system was functioning correctly, rather than relying on some inaccuracy in the simulator. However, the Tigon 2 has insufficient processing and memory resources to support more than about 100 Mbps when performing connection handoff. It was sufficient to verify correctness and to gather some information about system behavior. For a more thorough evaluation, the exact operating system and firmware used on the prototype were run within the Simics [14] full system simulator. Simics modules were implemented and/or modified to model the behavior of a network interface that supports connection handoff. The existence of a prototype allowed the simulator to be validated against systems with and without support for connection handoff, lending further credibility to the experimental evaluation of connection handoff. RiceNIC would have obviated the need for a second, simulation-based evaluation, as it has sufficient processing and memory resources to fully evaluate connection handoff in a real system.

# 4. EDUCATION WITH RICENIC

In addition to being a useful experimental tool for network server architecture research, RiceNIC can also be used to enable experiment-based education in computer architecture and networking courses. The complete and freely available RiceNIC reference design, which includes both hardware and software design files, provides a convenient baseline system and allows students to focus on implementing the specific architectural modifications in question, instead of building a functional NIC from scratch. This allows even a semester-long course to integrate hands-on engineering with classroom work.

To evaluate the flexibility of the RiceNIC design for education purposes, the baseline NIC was modified to include network address translation (NAT) services. Obviously, NIC-based NAT services are not novel, but their implementation does serve to show how RiceNIC firmware can be easily extended to implement additional services above and beyond basic networking. This is representative of a class project that could experimentally compare the tradeoffs between performing network address translation on the host processor or on the NIC.

RiceNIC firmware was modified to inspect each packet and perform NAT services in which all internal nodes can initiate connections to the external network, but external nodes can only initiate connections to specific ports that are forwarded to internal servers. All traffic not belonging to either connection type is dropped. A server with one RiceNIC and one conventional NIC can then operate as a fully functioning NAT firewall. RiceNIC is the interface to the external network and the conventional NIC is the interface to the internal network. IP forwarding is enabled on the host Linux system to forward all packets between the internal and external networks.

In this configuration, RiceNIC was able to sustain TCP stream throughput within 3% of the theoretical Ethernet limit for incoming and outgoing NAT traffic, using maximum-sized packets. The NAT processing code is running on the same PowerPC processor that performs all of the NIC management tasks, and that processor still has idle cycles remaining. In addition, the second PowerPC processor is completely idle and available for other tasks. This shows that RiceNIC firmware can easily be extended with new functionality and that substantial processing resources exist on the NIC for more advanced projects. While only the firmware was modified in this example, significant FPGA hardware resources are also available on the NIC, and some packet-inspection and processing tasks could be accelerated in hardware as a more advanced class project.

Experimental education uses for RiceNIC are not limited just to network interfaces, however. In conjunction with other freely-available academic tools, a whole reconfigurable and programmable network ecosystem can be constructed in the classroom. FPGA-based switches and routers have been created [13], including the active "NetFPGA" design from Stanford [4, 24]. These boards and programming kits are freely available for academic and research use.

The second generation system, NetFPGA-v2, provides a four-port Gigabit Ethernet PHY, a Virtex-II Pro FPGA with two PowerPC processors, 4 MB of onboard memory, and a 32-bit/33MHz PCI bus interface [24]. This board also provides high-speed serial links to connect several NetFPGA boards together. The system has been used in several project-based classes where teams of students work to design, construct, and test an internet router or smart Ethernet switch. Such a course allows students to gain hands-on experience with common Internet protocols. Because both FPGAs and embedded processors are available for use, students can experimentally explore the tradeoffs in performance and design complexity between implementing key IP router features in hardware and software [24]. In addition, the course also includes an open design portion where the students choose their own feature to design and test. The chosen projects span a broad spectrum of network topics, including the implementation of MAC-level encryption, a hardware firewall, VPN support, and man-in-the-middle SSH attacks [4].

The NetFPGA and RiceNIC projects can be used in a complimentary fashion. NetFPGA is well suited for use as a router or switch due to its 4 network ports and slower 33MHz interface with the PCI bus that limits communication to the host system. RiceNIC is better used as a NIC because it only has 1 network port and a faster (64-bit/66-MHz) interface with the PCI bus. The NetFPGA project includes reference designs for an IPv4 router and network switch, while RiceNIC provides a NIC reference design. Both systems could be used to build an entirely reconfigurable and programmable networking lab where all of the endpoints and routing fabric can be modified at will. Such a laboratory would easily allow for the experimental exploration of hardware/software design tradeoffs in network system architectures. It would also allow the implementation and experimental evaluation of new networking architectures at all levels of the spectrum, from a single device to a whole ecosystem of networked devices.

# 5. EXPERIMENTAL TOOLS IN NETWORK SERVER ARCHITECTURE

RiceNIC is a capable and flexible prototyping tool that can be used to evaluate future network server architectures.

Prior to RiceNIC, experimental network server work has relied on full-system simulation and/or prototyping with modern programmable NICs. These techniques have several drawbacks that are mitigated by the RiceNIC platform.

## 5.1 Simulation

In recent years the network server architecture community has embraced simulators to experimentally evaluate the performance of new and innovative techniques involving both hardware and software architectural changes [2, 3, 14, 19]. Simulation is often preferred because of the complexity, perceived or real, of implementing a new architecture in hardware. Relying solely on a simulator to accurately predict future performance trends, however, is problematic. Past research has shown the strengths and weaknesses of simulation techniques in both computer architecture and, more specifically, network server architecture [5, 6, 20].

When simulating an existing computer system with known behavior, the key elements affecting performance can be determined and modeled with high fidelity, and less significant factors approximated to reduce simulation time and complexity. Previous research showed that a simulator can be an accurate predictor of network performance trends, but only if it models all of the important factors that contribute to system performance [6]. In simulations of new network architectures, however, it is often hard to predict in advance which factors contribute to latency in a real system, and which can be neglected or approximated. An experimental prototype can be used to validate the simulator, correct any errors or omissions in the design and establish an accurate baseline. From this baseline, incremental architectural changes can be proposed and experimentally tested with high confidence that the simulated results are indicative of real system behavior. If the experiment deviates too far from the baseline design, however, the initial simulator design assumptions and simplifications can become invalid, and the accuracy of the simulation will suffer.

Modeling a network interface and its interactions within a complete computer system challenges simulation designers to extend their reach beyond the traditional user application, processor and memory models. The performance of a NIC is highly dependent on the complex interactions between many asynchronous components within the system, including the processor, memory, I/O interconnect, and the NIC itself. Memory latency, I/O bus contention, and interrupt latency are only some of the key factors which significantly affect total system performance. None of these factors can be modeled with fixed latency or other constant parameters, but they all need to be simulated with high fidelity [3]. In addition to the hardware, a full software environment needs to be simulated, including both application and operating system code, where the bulk of network processing is performed. Finally, in many network architecture experiments, multiple whole systems need to be simulated along with the network connecting them together.

Even the most recent full system simulators, such as M5 [3] and Simics [14], avoid using precise models whenever possible, due to the implementation complexity and computational time required for full simulation. Full system simulation can be orders of magnitude slower than running an identical benchmark on real hardware. Instead, simple generic component models are used and tuned to achieve the correct bandwidth and latency.

In the case of the M5 network simulator, the authors were able to tune the simulator to achieve network throughput results within 15% of the actual hardware being modeled [20]. To achieve this level of accuracy, however, required an iterative tuning process involving precise adjustments to the memory system performance, processor TLB latency, and other complex systems. Even after tuning the simulator, significant differences remained. For example, the NIC transmit interrupt rate was twice as high as real hardware, which increased the processor overhead of interrupt-handler routines and decreased the simulated throughput. If a real test machine had not been available to compare the simulator against, this effect might never have been detected.

TCP performance can be sensitive to side-effects of common simulation techniques. Because it is infeasible to simulate complete runs of real-world workloads, experiments may be conducted by first using a fast functional simulator during the "warm-up" stage of a test, and then running a highly detailed (and much slower) microarchitecture simulator and taking a short data sample assumed to be representative of the average system performance. However, the TCP self-tuning policies can interfere with this technique [7]. TCP will tune itself to unreasonably high throughput with the fast functional model, and will suddenly become processor-limited once switched to the complete simulator. The effect on TCP behavior is similar to sudden network congestion, and TCP will immediately begin reducing its bandwidth consumption to avoid packet loss and reach a new steady-state. Capturing performance figures during this period of unstable network throughput can lead to misleading results. The simulation time required for TCP to complete its tuning process can vary widely, from 10 to 150 million cycles in a system with zero network delay, to even longer in systems with real-world network delays. This forces researchers to either perform very long-running simulations in order to capture accurate results, or carefully characterize their workloads and configurations in order to cut the TCP tuning time to a more reasonable simulation length.

Furthermore, the behavior of TCP is strongly timing dependent and is very sensitive to the performance of the simulated system [7]. This is because the TCP algorithms use the current network performance as a key to controlling future network communication. If packets are received from the network faster than they can be processed, the receiver buffers them until memory is exhausted, at which point packets are lost. When the transmitter detects these dropped packets, it retransmits the missing packets and also decreases its transmission rate and transmit buffer size. Thus, small timing variations, possibly due to small performance modeling omissions or simplifications in the simulator, can significantly alter the TCP execution path and thus its behavior and performance.

Such timing-dependent effects are not limited to just the TCP stack. Modern NICs attempt to aggregate multiple packets together to be processed by the operating system with a single interrupt, thus amortizing the high overhead of

a hardware interrupt across multiple packets. Small changes in packet arrival and departure time can affect the rate at which the NIC interrupts the host operating system for service, and also vary the amount of packet processing that is done on a per-interrupt basis. Variations in the interrupt rate and the amount of aggregated networking work done per interrupt can significantly affect the processing efficiency and performance of heavily-loaded systems. Thus, accurate simulation is difficult due to the sensitivity of networking systems to small timing effects.

Revolutionary improvements in the field of computer I/O architecture will likely require large-scale modifications to both the hardware and software interfaces at the same time. Changes of this magnitude, however, will alter the nature of asynchronous communication in the system, challenging the use of simulation-based experimentation. They may bring previously second-order performance effects to the forefront, potentially skewing the simulation and providing misleading results. Even if the baseline simulator has been validated, the new I/O architecture also must be validated to ensure accurate results. Thus, simulation of network server architecture for both current and future research efforts has inherent challenges that are difficult to overcome.

## 5.2 Prototyping

The complexity of modern I/O architectures and the difficulties of simulating them motivates the use of prototyping for experimental evaluation. Therefore, many research projects involving the network subsystem construct prototypes using software programmable NICs, either for Ethernet or specialized interconnects such as Myrinet or Infiniband.

The Tigon 2 [1] is a programmable, full-duplex, gigabit Ethernet NIC that has been frequently used for networking research. The NIC contains two in-order 88MHz RISC processors, up to 2 MB of SRAM, and a 64-bit/66 MHz interface with the PCI bus. Device drivers, firmware, and tools were publicly distributed for the Tigon. Projects that made use of the Tigon 2 include: user-level message passing [22], firmware parallelization for 2-CPU NICs [9, 23], network interface data caching [12], and user-level network access [18]. In all cases, the authors extracted as much performance from the hardware as possible; however, some limitations in the Tigon 2 NIC are difficult to bypass. First, the memory and computational resources are extremely limited by modern standards, making aggressive experimentation for performance analysis purposes challenging. Second, the shared hardware units, such as the MAC and DMA, have no concept of concurrency and require external synchronization between the processors. Third, parallelization efficiency could be improved since the Tigon 2 only implements a single semaphore for CPU synchronization.

In addition to Ethernet NICs such as the Tigon, other academic research projects have used specialized Myrinet or Infiniband NICs. These recent NICs have faster embedded processors and greater memory resources than the Tigon. Projects involving payload caching on routers and firewalls [26] and NIC-based intrusion detection [17], for example, have used these specialized NICs as prototypes. However, these projects are actually targeted at, and best suited for, commodity Ethernet networks in commercial applications, and not high performance supercomputing networks. Building a prototype using one interconnect and proposing a final system with a different interconnect poses challenges for researchers who must reliably map performance results between the two standards.

Due to the complexity of simulation, as discussed in Section 5.1, prototyping is often preferred for experimental research into novel network server architectures. However, existing prototyping solutions also have their drawbacks. RiceNIC offers a promising alternative for future network interface prototyping. RiceNIC can enable a wider range of innovations, as it provides more computation and memory capacity than existing programmable NICs. Furthermore, RiceNIC also provides an additional level of configurability and expandability with the availability of significant FPGA resources. This should enable efficient prototype implementations of advanced hardware/software innovations for future network servers.

# 6. CONCLUSIONS

Networking is pervasive in modern computer systems, and the efficiency of network processing is critical to many server applications. Experimental research that involves modification to the hardware and software of the network subsystem is becoming increasingly important. RiceNIC, a programmable and reconfigurable Gigabit Ethernet network interface, enables the efficient creation of experimental hardware prototypes. RiceNIC is an open platform for network interface research. The Avnet development board is commercially available, and the FPGA configuration and supporting software, including firmware and device drivers, are provided freely for research and education.

Everything on RiceNIC is easily modifiable and is oriented towards experimentation. The serial console makes RiceNIC a friendly platform for research and education. The NIC can easily display status information and the user can interactively control the NIC. The capabilities of the platform also provide the opportunity for other tools that are commonly found only on general-purpose systems, such as a timer-based statistical profiler. Significant computation and storage resources are provided that are largely unutilized when performing the basic tasks of a network interface. Over half of the processing and memory resources on the Avnet board are available for customization. This makes RiceNIC an ideal platform for experimental research into advanced networking architectures that require new services of the network interface.

The case studies of the previous sections highlight the usefulness of RiceNIC in both research and education settings. The RiceNIC prototype for concurrent direct network access in virtual machines was able to establish the full performance potential of the design and experimentally determine key configuration parameters such as buffer sizes. Using the Gigabit Ethernet RiceNIC produced a more convincing demonstration than approaches based on existing network interfaces or simulation because it is much closer to an actual system implementation. In another case study, the NAT firewall was easily implemented by simply modifying the firmware, and showed how such tasks could easily be done in a classroom setting.

Experimental research involving the network system is not only possible, but is essential. The complex, asynchronous interactions among system components and other external systems demand high performance prototyping for accurate experimental research. RiceNIC, and other tools like it, are critical for the development and understanding of future computer systems.

To learn more about RiceNIC, please visit:
http://www.cs.rice.edu/CS/Architecture/ricenic/

## 7. ACKNOWLEDGMENTS

The authors thank Paul Willmann, David Carr, Alan Cox, Aravind Menon, and Willy Zwaenepoel for their work on Concurrent Direct Network Access utilizing RiceNIC. This project is supported by gifts from Advanced Micro Devices and Xilinx, and grants from the Los Alamos Computer Science Institute and the National Science Foundation under grant No. CCF-0546140.

## 8. REFERENCES

[1] Alteon Networks. *Tigon/PCI Ethernet Controller*, August 1997. Revision 1.04.

[2] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[3] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: Modeling networked systems. *Micro*, 26(4):52–60, 2006.

[4] M. Casado, G. Watson, and N. McKeown. Reconfigurable networking hardware: A classroom tool. In *Proceedings of the Symposium on High Performance Interconnects*, Aug. 2005.

[5] R. Desikan, D. Burger, and S. W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the International Symposium on Computer Architecture*, June 2001.

[6] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich. FLASH vs. (simulated) FLASH: closing the simulation loop. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[7] L. Hsu, A. Saidi, N. Binkert, and S. Reinhardt. Sampling and stability in TCP/IP workloads. In *Proceedings of the Workshop on Modeling, Benchmarking, and Simulation*, June 2005.

[8] H.-Y. Kim, V. S. Pai, and S. Rixner. Increasing web server throughput with network interface data caching. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[9] H.-Y. Kim, V. S. Pai, and S. Rixner. Exploiting task-level concurrency in a programmable network interface. In *Proceedings of the Symposium on Principles and Practices of Parallel Programming*, June 2003.

[10] H.-Y. Kim and S. Rixner. Connection handoff policies for TCP offload network interfaces. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, Nov. 2006.

[11] H.-Y. Kim and S. Rixner. TCP offload through connection handoff. In *Proceedings of EuroSys*, Apr. 2006.

[12] H.-Y. Kim, S. Rixner, and V. Pai. Network interface data caching. *IEEE Transactions on Computers*, 54(11):1394–1408, Nov. 2005.

[13] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable network packet processing on the field programmable port extender (FPX). In *Proceedings of the International Symposium on Field Programmable Gate Arrays*, Feb. 2001.

[14] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hállberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[15] Microsoft Corporation. *Scalable Networking: Network Protocol Offload – Introducing TCP Chimney*, Apr. 2004.

[16] J. Mogul, L. Brakmo, D. E. Lowell, D. Subhraveti, and J. Moore. Unveiling the transport. *Computer Communication Review*, 34(1):99–106, 2004.

[17] M. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, and D. Panda. Towards NIC-based intrusion detection. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2003.

[18] I. Pratt and K. Fraser. Arsenic: A user-accessible Gigabit Ethernet interface. In *Proceedings of IEEE INFOCOM*, Apr. 2001.

[19] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: The SimOS approach. *Parallel Distributed Technology*, 3(4):34–43, 1995.

[20] A. G. Saidi, N. L. Binkert, L. R. Hsu, and S. K. Reinhardt. Performance validation of network-intensive workloads on a full-system simulator. In *Proceedings of the Workshop on Interaction between Operating System and Computer Architecture*, Oct. 2005.

[21] J. Shafer and S. Rixner. A reconfigurable and programmable gigabit ethernet network interface card. Technical Report TREE0611, Rice University, Dec. 2006.

[22] P. Shivam, P. Wyckoff, and D. Panda. EMP: Zero-copy OS-bypass NIC-driven gigabit ethernet message passing. In *Proceedings of the Conference on Supercomputing*, Nov. 2001.

[23] P. Shivam, P. Wyckoff, and D. Panda. Can user-level protocols take advantage of multi-CPU NICs? In *Proceedings of the International Parallel and Distributed Processing Symposium*, Apr. 2002.

[24] G. Watson, N. McKeown, and M. Casado. NetFPGA: A tool for network research and education. In *Proceedings of the Workshop on Architecture Research using FPGA Platforms*, Feb. 2006.

[25] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, Feb. 2007.

[26] K. Yocum and J. Chase. Payload caching: High-speed data forwarding for network intermediaries. In *Proceedings of the USENIX Technical Conference*, June 2001.