The *END*: An *E*mulated *N*etwork *D*evice for Evaluating Adapter Design[*]

Atri Indiresan, Ashish Mehra, and Kang G. Shin

Real-time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122
313-763-0391 (voice); 313-763-4617 (fax)
{*atri,ashish,kgshin*}@*eecs.umich.edu*
http://www.eecs.umich.edu/RTCL/atri

1 Introduction

Recent advances in high-speed networking have made possible distributed applications with various requirements on end-to-end communication, including minimum bandwidth, bounded delays, delay jitter and packet loss rates. End-to-end communication performance depends not only on the underlying networking technology, but also on the end-host operating system and the interface between the host and the network. As network speeds increase, the performance bottleneck tends to shift to the end host, in particular to the hardware and software components of the host's communication subsystem. The design of the network adapter, and the division of functionality between the adapter and the host communication software, can have a significant effect on the performance delivered to applications.

To study the hardware and software issues in network interface design, we need to be able to study various algorithms and interfaces, and observe and analyze the hardware and software overheads under different traffic conditions. Clearly, this is part of the design process, and needs to be done before building the adapter card, since these are typically hard to modify.

We propose network device emulation as a mechanism to study the hardware-software interface for communication, and to help design network adapters that integrate well with the host operating system and applications. In particular, we propose to study the performance of adapter designs via an *E*mulated *N*etwork *D*evice (*END*) that interfaces to a real communication protocol stack on the host. The *END* can emulate all the operations required of a network adapter, without having to interface to a real network. *END* uses a synthetic network model that acts as a sink and source of traffic. Since *END* can construct arbitrary network models, we can study networks with a wide range of parameters. This is a useful approach to evaluating adapter design tradeoffs since it allows us to model a network while accounting for interactions with a target host system.

The main goal of this paper is to propose device emulation as an effective evaluation technique to study the performance considerations and tradeoffs in network interface design. While we focus on issues in data transmission, a companion paper [1] presents further details of *END*, including data reception issues, as well as implementation and evaluation results. Section 2 motivates emulation as a performance evaluation tool. Section 3 describes the issues involved in designing network adapters, and Section 4 shows how *END* may be used to study them. Section 5 describes some applications of network device emulation, and we conclude in Section 6 with a brief summary of the current status of *END*.

2 Why Emulation?

In order to design a network adapter that meets certain performance requirements, one must study the impact of various design parameters in a realistic set-

[*]The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895 and the Office of Naval Research under Grants N00014-91-J-1115 and N00014-94-1-0229. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

ting, i.e., when the network adapter interacts with the communication software on a target host platform. Building and testing hardware, and interfacing to the operating system, is time-consuming and expensive. Most adapters do not allow on-board firmware to be modified or programmed to experiment with different design tradeoffs. More importantly, more often than not, the hardware engineers designing network adapters are far removed from the concerns of those writing communication software, resulting in a design poorly integrated with the rest of the host operating system.

Several techniques may be used in designing and evaluating network subsystems. Mathematical models are typically used to evaluate the queuing behavior of network traffic. However, such models are usually simplified to make the analysis tractable, and they rarely account for system overheads of interrupt handling, context switches, etc., encountered in practice.

Another technique is simulation, which has several significant advantages [2], including ease of development, flexibility and versatility. Since a simulator is built in software, it can be readily modified and augmented to test new features and interfaces. Simulators are usually easier to build and cheaper than real systems. They can model "ideal" systems that are impossible to build, e.g., an infinitely fast network. However, a simulator is typically an artificial device, i.e., no real system components are involved. Detailed models of system components are necessary for accurate simulation. These could include hardware components like CPU, I/O buses, memory, caches and interrupts, and software components including the operating system, communications protocol stack and the applications. Further, there are complex and subtle interactions between these components that also need to be modeled. In general, not only is it difficult to provide accurate models, but, the greater the detail of the models (and hence, greater accuracy), the slower the operation of the simulator.

Exceptions to this do exist in approaches that execute actual software under control of the simulator [3]. This not only enables the simulator to get accurate timings for execution of code segments, but also greatly speeds up the simulation. However, while sufficient to study the performance of software components, such approaches are not sufficient when hardware-software interaction and concurrency needs to be captured as well. For instance, execution of a code segment could be affected by interrupts, which could affect execution order, and hence, the order of transmission of data.

In contrast to a simulator, an emulator simulates the behavior of a system component, and interfaces to a real system. This implies that the emulated component must operate in real-time, and give the rest of the system the impression that it is interacting with a real subsystem. In our case, *END* emulates a network adapter and interfaces with the target host, giving the impression that the host communication software is communicating with a real network. This has several significant advantages over simulation. Interfacing a device emulator to the target host allows adapter design tradeoffs to be evaluated in the presence of applications, operating system overheads, interrupts, etc. Device emulation shares the advantages of simulation in that it is a flexible technique that allows rapid design and evaluation of various interfaces and adapter design policies and/or algorithms. This helps identify design limitations and bottlenecks early in the design cycle. Further, since device emulation is carried out on the target platform, it allows development and testing of host operating system software, and rapid integration of the actual hardware device when it becomes available. Emulation suffers from one significant drawback compared to simulation. Since the emulator runs in real time, any overhead for instrumentation and observation is real, and affects performance, unlike in a simulator,

Other Emulators: Emulation has been used in different ways to evaluate networks. *Hitbox* [4] injected delays in Ethernet links connected in a point-to-point mode to emulate the delay and throughput of Wide Area Network links. It emulates the throughput and delay behavior of the network, as long as it is not faster than the Ethernet. Blair *et al.* [5] used transputer based point-to-point connections to emulate the protocols and behavior of a FDDI network. *END*, on the other hand, emulates in detail not only the throughput and delay of *individual components* of a network adapter, but also the protocols and algorithms running on the adapter. Downloading protocols to communication cards have been suggested as a means of enhancing performance [6]. *END* may be used to evaluate and select protocols that may be executed on adapter cards, even before the design and implementation of the hardware is complete.

3 Issues in Network Adapter Design

Figure 1 illustrates the architecture of a typical network adapter. There are five basic components that comprise this architecture: the host/adapter interface[1], the data transfer control, transmis-

[1]Note that most network adapters are accessed by the host via the system I/O bus [7].

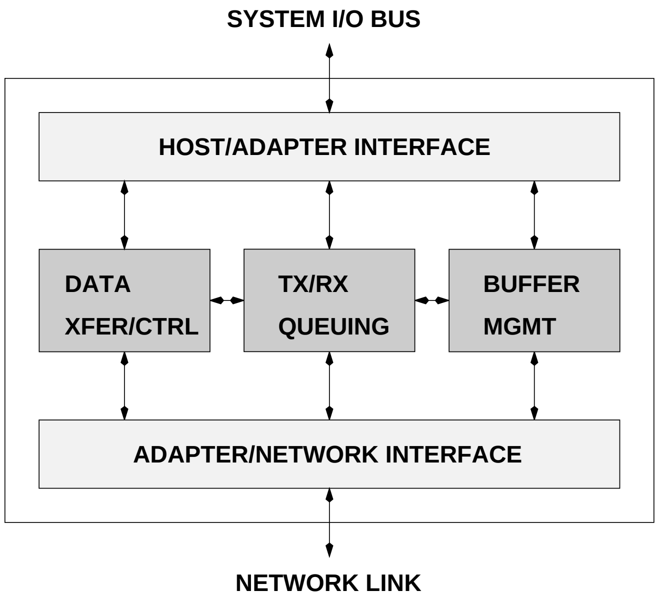

Figure 1: Architecture of a generic network adapter.

sion/reception queuing, buffer management, and the adapter/network interface. For efficient and flexible implementation, an adapter typically employs one or more general-purpose microprocessors, as well as custom hardware, under the control of the adapter firmware.

Adapters can be quite complex, depending on their performance goals and the underlying network technology. Selection of the design parameters listed above is a major determinant of network performance. *END* permits a designer to explore a cross-product of the design space, and make appropriate choices.

3.1 Adapter Interfaces

The Host-Adapter Interface: The host/adapter interface is typically in the form of a device driver that runs on the host, and a companion "host driver" on the adapter. The two drivers exchange information via command-response mailboxes or queues, and may synchronize their operations either via interrupts, polling, or some combination of the two. Interrupts allow rapid response to asynchronous events, especially those that occur infrequently, but incur greater overhead than polling.

The Adapter-Network Interface: The adapter transmits (receives) data to (from) the network medium by copying data from (to) its buffers to (from) the network under control of the medium access protocol of the attached network. It may use either Programmed I/O (PIO) or DMA for this operation. Since DMA uses custom memory copying hardware, it not only is typically faster than PIO, but also keeps the CPU free to be used for other activities. However, due to initial set up costs of DMA, it may not be suitable for small memory transfers.

3.2 Adapter Internals

There are three main modules providing the internal functionality of the adapter: data transfer control, transmission/reception queuing, and buffer management. In addition, it may provide services such as segmentation and reassembly, CRC computation, encoding and decoding, encryption and decryption, etc.

Data Transfer Control: Data may be transferred between host memory and adapter buffers via DMA or PIO. The performance tradeoffs are similar to those discussed above for the adapter-network interface.

Tx/Rx Queuing: Once packet transmission is initiated, or a packet arrives from the network, the adapter must queue the packet until it can either be injected into the network (transmission) or received by the host (reception). Queuing policies and mechanisms may depend on the expected traffic mix. For example, best-effort adapters may simply use first-in-first-out (FIFO) queuing of packets with deep pipelining of operations on the adapter. This delivers high throughput by exploiting the overlap between, say, DMA of one packet, and transmission of another, and keeps the queuing overheads low.

Buffer Management: Adapters provide buffers as a staging area for packet transmission and reception. The buffers may reside either in adapter or host memory. In the former case, the adapter must provide buffer management policies, as well as handle buffer overload conditions correctly. The adapter may also exercise partial control over packet buffers on the host, and may allocate and free them for data transfers.

4 Network Adapter Emulation

We now examine how emulation may be used in the study of network adapters and networks.

4.1 Host View of the Network

In order to accurately emulate a network, *END* must export to the host the same view as that exported by a real adapter. At the very minimum, *END* must be capable of being integrated with the host system and give the applications and host operating system the impression that there is a real network supporting a given interface with specific performance characteristics. An application's view of the network is via the operating system interface and the communication protocol stack; since *END* is intended to run on the target system, these need not be changed. The device specific interface to a network, on the other hand, is implemented on the host as a device driver.

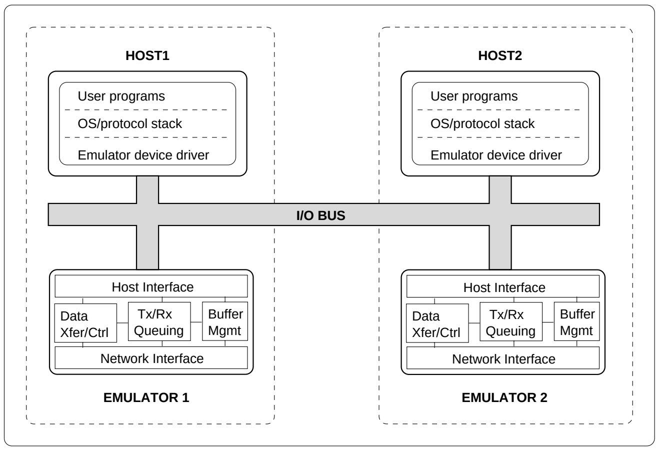

Figure 2: *END*-based device emulation architecture.

END should interact with this device driver the same way as a real device would – handling commands, issuing responses, and synchronizing with the host using interrupts or polling. In addition, it must also be seen as a sink (for transmission) and source (for reception) of data. When an application sends data, the adapter acknowledges successful transmission of the data after a delay (determined by the size of the data buffer, the data transfer rate, and the system and network load), and allows the application/host to reuse the data buffer. As long as the application does not expect an acknowledgment from its peer receiving application, all it sees is data being transmitted at a certain rate. A transmitting application will not be able to distinguish between *END* and a real network as long as *END* captures this timing behavior.

4.2 Emulator Components

Figure 2 illustrates our emulator-based architecture for studying protocol and performance issues in adapter design. Each network "node" corresponds to a host processor board and an emulator processor board (running *END*) on the same (system) I/O bus. Since delays are used to model network transmissions, only one network node suffices for transmission experiments. Two-way communication, on the other hand, is accomplished via two or more such nodes on the same I/O bus, with the I/O bus serving as the communication medium. This may be used to verify the correctness of protocols. However, the bandwidth of the I/O bus can be a limiting factor in the speed of the network being emulated. Further, unlike full duplex network connections, contention for the shared bus would affect performance. If the actual data being transmitted is not important, it is enough that emulator nodes just exchange packet header data. If the header lengths are small compared to the data, this not only allows emulation of faster networks, but also reduces contention for the bus.

Host Emulator Driver: The host node is precisely the target host. Communicating user applications send and receive data via the protocol stack in the operating system. At the bottom of the protocol stack is the host emulator driver for the target network adapter. Since *END* and a real device present the same interface to the host, this allows a developer to implement and test a complete device driver even while the network adapter is being designed or implemented. It also ensures that observed performance of the driver is the same for the emulator as for the real adapter.

Network Adapter: The network adapter node is a general-purpose processor board, with a CPU and memory. Since its main function is to handle data movement rapidly and efficiently, the adapter node needs a minimal executive to handle interrupts and provide concurrency. As discussed earlier, pure packet transmission can be modeled as a transmission-complete notification after a suitable delay. This can be reasonably accurate, except that since actual data is not transferred, the host CPU does not suffer the overhead of cycle stealing while DMA copies data.

Time Services: *END* needs time services to represent delays corresponding to data movement. This requires a high resolution event server to register delays, and notify the client when the time expires. The notification could be either an interrupt (modeling, for example, a transmission-complete interrupt), or simply setting a completion flag that the adapter can poll.

5 Emulation Applications

Emulation has several applications, including performance evaluation, verification of protocols, and performability analysis. We discuss some examples below.

5.1 Design for QoS

Quality-of-service (QoS) requirements, which may include minimum bandwidth, bounded delays, delay jitter and packet loss rates, are often required on connections between peer applications. Though various flow control and queuing schemes have been suggested to provide these (per-connection) guarantees [8], we have shown that implementation/system overheads [9], CPU scheduling [10], and the division of services between the host and adapter [1], greatly influence performance in ways that are often not obvious. *END* permits rapid implementation and evaluation of various policies and algorithms. See [10] for an example of QoS evaluation using simple emulation techniques, and experiments using *END* are described in [1].

5.2 Performability Analysis

END may be used to verify the correctness of protocols, and study their behavior in the presence of faults. Since the entire behavior of an adapter is being emulated, it is easy to corrupt, delay, drop or replicate data in a controlled manner, and observe end-to-end protocol behavior. Further, since *END* may be configured with multiple interfaces (host-adapter and adapter-network), it can emulate redundant connections between network nodes, and evaluate the behavior of fault-tolerant architectures and protocols in the presence of faults.

6 Conclusions and Future Work

In this paper we have proposed a framework for exploring issues in adapter design via device emulation. We have designed and implemented an initial prototype of *END* [1], which is capable of representing the functionality of a network adapter at different levels of detail. Further, network protocols may be divided between the host and the adapter in different ways. Since it interfaces with a real host, *END* operates in real-time and can be quite effective in evaluating network adapter and host-adapter interface designs early in the design cycle. We believe that the hardware-software codesign facilitated by *END* can help design network adapters that integrate well with the host architecture and operating system.

At present, *END* allows a designer to implement adapter functions by writing the code from scratch. To make *END* a more flexible and easy-to-use performance evaluation tool, we are developing a taxonomy of adapter and host features and configurations. This taxonomy will allow us to develop libraries of generic and specialized models of adapter components that may be combined to rapidly realize and evaluate a desired adapter design. Further, since *END* is a software module emulating hardware, it will also be possible to model faulty behavior in the communication subsystem, to evaluate dependability in addition to performance. Finally, we are exploring the issues involved in applying emulation to evaluate other I/O devices such as disk drives.

References

[1] A. Indiresan, A. Mehra, and K. Shin, "The *END*: Exploring QoS issues in adapter design via an *E*mulated *N*etwork *D*evice," submitted for publication, June 1996.

[2] R. C. Bedichek, "Talisman: Fast and accurate multicomputer simulation," in *Proceedings of Sigmetrics 95/Performance 95*, pp. 14–24, May 1995.

[3] L. S. Brakmo and L. L. Peterson, "Experiences with network simulation," in *Proceedings of ACM Sigmetrics 96*, pp. 80–90, May 1996.

[4] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP vegas: Emulation and experiment," in *Proc. of ACM SIGCOMM*, pp. 185–195, October 1995.

[5] G. Blair, A. Campbell, G. Coulson, F. Garcia, D. Hutchison, A. Scott, and D. Shepherd, "A network interface unit to support continuous media," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 264–275, February 1993.

[6] R. K. Budhia, P. M. Melliar-Smith, L. E. Moser, and R. Miller, "Higher performance and implementation independence: Downloading a protocol onto a communication card," in *Proc. of the Intl. Conf. on Comm.*, June 1995.

[7] K. K. Ramakrishnan, "Performance considerations in designing network interfaces," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 203–219, February 1993.

[8] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 122–139, January 1994.

[9] A. Mehra, A. Indiresan, and K. Shin, "Resource management for real-time communication: Making theory meet practice," in *Proc. of 2nd Real-Time Technology and Applications Symposium*, June 1996.

[10] A. Mehra, A. Indiresan, and K. Shin, "Structuring communication software for quality-of-service guarantees," in *Proc. of 17th Real-Time Systems Symposium*, December 1996.