The *x*-chip: An Experiment in Hardware Demultiplexing

Mary L. Bailey Michael A. Pagels Larry L. Peterson
Department of Computer Science
The University of Arizona
Tucson, Arizona 85721

1 Introduction

An important issue in the design of host-network interfaces is how much protocol processing should be done in hardware (i.e., on the network device) and how much should be done in software (i.e., on the host). This issue is becoming increasingly important as improving network performance puts more and more stress on the end host. As a result, there have recently been several efforts to experiment with different points in the design spectrum: (1) Nector's communications adaptor board (CAB) [1]; (2) Network Adaptor Board (NAB) [2]; and, (3) Aurora gigabit network testbed [3]. Each of these architectures implements one or more protocols on the network device—in the case of CAB, it is an entire protocol suite; in the case of NAB, it is a single high-level (transport) protocol; and in the case of Aurora, it is a single low-level protocol.

This paper introduces another point in the design spectrum: the *x*-chip. Rather than implement one or more protocols on the network device, the *x*-chip implements a collection of *protocol*-independent *functions* on the network device. The *x*-chip does not replace the software implementations of protocols; it just provides hardware assist to these protocols. The functions provided are simple enough that they can be implemented by specialized chips; general purpose processors are not used in the design.

The *x*-chip is an outgrowth of our work on the *x*-Kernel—a software architecture that supports protocol implementations [4]. One of the design philosophies of the *x*-Kernel has been to identify tasks that are common to all protocols, and to provide general tools that support these tasks. The *x*-chip takes this philosophy one step further by moving certain aspects of the support tools into hardware, with the the ultimate goal of vertically integrating the network device and the network software infrastructure.

2 Design Rationale

Our decision as to what functionality to move into the network device revolves around three factors: processor cycles, memory cycles, and resource scheduling. In the case of processor cycles, the goal is to (1) minimize the overhead involved in servicing network interrupts, and (2) minimize the number of times the processor has to inspect/manipulate the data contained in each network packet. In the case of memory cycles, the goal is to minimize the number of times a network packet is moved from one buffer to another. Clearly, minimizing the number of times packet data is picked up and put down is closely related to minimizing the processor cycles devoted to inspecting/manipulating the data. Finally, we recognize the need to distinguish among different classes of packets, so that policy decisions can be made with regard to allocating various resources (e.g., processor, display, outgoing network connections) to those packets.

We have identified three key functions that we believe should be provided in the network device: demultiplexing, reassembly, and header/data separation. Intuitively, moving these functions into hardware takes advantage of the fact that network packets arriving at a particular host exhibit *temporal locality* (a sequence of packets addressed to the same process arrive in a brief period of time) and *spatial locality* (a small, slowly changing set of processes are receiving packets at any given time)[5].

2.0.0.1 Demultiplexing.

An important function performed by network protocols is the multiplexing of independent data streams onto a single outgoing (possibly logical) channel, and the demultiplexing of packets arriving on a channel into distinct data streams. The demultiplexing half of the problem has serious consequences on performance [6]. In a network architecture with multiple protocol levels, a demultiplexing decision is made at each level, meaning that multiple protocols must process the packet before the ultimate data stream can be determined [7].

The *x*-chip assists with this function by providing a facility that demultiplexes each incoming packet into different buffer pools based on certain properties of the packet. Note that the hardware only selects a receive buffer into which the packet is placed; each protocol still processes the packet. The protocols, however, affect only the flow of control through the protocol stack; the packet is not moved from the receive buffer.

2.0.0.2 Reassembly.

A common function performed by many protocols is fragmenting a large outgoing message into multiple network packets on the sending host, and reassembling multiple incoming packets into a single message on the receiving host. As with multiplexing/demultiplexing, the receiving side of the function is the most critical to performance. It would be advantageous if the network device could place packets belonging to the same message into consecutive buffers. This would eliminate the need to copy individual fragments into a contiguous buffer and implies that the device would have to interrupt the host only once for the entire message, rather than for each fragment/packet.

The *x*-chip provides a facility to place related packets into successive offsets of the same buffer, and to interrupt the host only when a collection of related packets has been received. The chip works in concert with the protocol that provides fragmentation/reassembly to determine the right circumstances under which an interrupt should be generated.

2.0.0.3 Header/Data Separation.

One of the conclusions we came to when implementing protocols in the *x*-Kernel is that it is useful for protocols to be able to distinguish between the user data and the set of headers that have been added to the message. The *x*-chip is designed to distinguish between these two parts of the message, and to scatter the incoming packet over multiple header and data buffers.

Most recent research on host interface design is being done in the context of some new (high-speed) network technology; e.g., the Aurora project is being built around a 622Mbs ATM network. In such context, two things generally happen: (1) it is possible to completely re-define the protocols being used, and (2) such efforts tend to focus on the network-device aspect of the problem, and seldom investigate the relationship between the network device and the host's software architecture.

In contrast, the *x*-chip is being designed independent of any particular networking technology. Put another way, we envision that it is possible to integrate the *x*-chip onto any network device. We believe that our approach has two desirable consequences: (1) we will attempt to find general solutions that work across a wide range of protocols, and (2) we will focus on the vertical integration of the network device and the host's software architecture.

3 The *x*-chip

The *x*-chip is designed to facilitate the three functions discussed above: demultiplexing, reassembly, and header/data separation. It is intended to be an experimental platform for investigating: (1) the impact of hardware assistance upon these key protocol functions; (2) the impact on hardware design, performance and cost; (3) protocol design and performance; and (4) operating system design and performance. The *x*-chip is intended to be protocol independent, supporting a wide range of protocols. The *x*-chip will be programmable to support different protocols and multiple applications in distinct address spaces. However, since many applications typically use similar protocols, we are providing optimizations for this common case. The *x*-chip will also support two types of host interrupts, so that there is flexibility in experimenting with different reassembly strategies. We envision a production quality *x*-chip to be a subset of a network controller chip, as opposed to an additional network coprocessor.

The *x*-chip is composed of two parts, a matcher and a router. The matcher determines the type of the incoming data stream by comparing specific fields in the packet with packet-type patterns the user has stored in the chip. The router separates the header and data information and computes the actual destination memory addresses. Additionally, the router is responsible for interrupting the processor when reassembly is "complete".

Packets enter the *x*-chip as a byte stream. As this byte stream passes through the matcher, it determines whether the current packet matches any of the packet-type patterns which have been stored on the chip. Packet types are identified by comparing a set of byte stream <*offset,value*> pairs which uniquely identify the packet type. If there is a match, then the packet is partitioned according to the "scatter" information stored in the router for this packet type. These byte stream partitions are then output together with the appropriate memory addresses for placement in primary memory. The router provides two methods to generate interrupts, providing interrupts on specific packet types or after the completion of a group of packets.

We have determined that for local area networks it is sufficient to inspect the first 50 bytes. This will accommodate typical TCP/IP packets, the bulk of ethernet LAN and WAN traffic, and should be sufficient for experimentation. In order to experiment with data scattering, the router will provide a small number of scatter fragments. We plan to implement the chip so that external programming can occur simultaneously with matching and routing to maximize chip throughput. Measurements of ethernet LAN and WAN traffic[5], [8] describe a preponderance of TCP and UDP packet types. We can take advantage of this bias by providing two forms of matching templates. In the first, a number of patterns share a common subset of <*offset,value*> pairs, and are differentiated by additional unique pairs. In the second there is no sharing of <*offset,value*> pairs, all are unique.

Additionally, we would like to experiment with a wide range of packet type patterns to provide experimentation with different protocols, including very long patterns. Thus we are planning to implement the *x*-chip as two chips, a matcher chip and a router chip. These chips will be constructed so that multiple chips can be linked "serially" expanding the range of the byte offset values. Similarly, chips will be designed so that we can link additional chips in "parallel" to extend the number of patterns which can be supported.

4 Analysis

In order to evaluate the usefulness of the *x*-chip, we need not only to evaluate the speed and cost of the hardware, but also the impact the *x*-chip will have in network protocols and operating systems.

4.0.0.1 Protocol and Operating System Experiments.

The most basic experiment using the *x*-chip prototype is to investigate how useful the chip is for general network protocols. This involves (1) demonstrating that it is feasible to use the *x*-chip in conjunction with software protocols, (2) quantifing the performance impact on protocols which are implemented using the *x*-chip, and (3) examining new implementation directions that may be required or suggested from this functional separation.

The structure of the *x*-Kernel provides us with a means, the *x*-Kernel's message/buffer management systems (message tool), to add support for *x*-chip functionality without the necessity of modifying existing protocol code. The message tool's scatter and gather functions become verification routines that only move data in the unlikely case that packets have arrived out-of-order. Our first experiment will be to verify that code modification can be limited to the message tool. In this way we are able to leverage off of prior *x*-Kernel development with its support of numerous network protocols, limiting our initial software development effort. We will experiment with message tool modifications and the modifications they induce within the protocol code in order to make more efficacious use of the *x*-chip's functionality.

Another issue in protocol design is how to deal with protocols that are sometimes supported by the chip and other times are not supported. Through modifications to the infrastructure's message handling abstractions our goal is to construct protocols such that the optimizations provided by the hardware are obtained whenever possible without any overt changes to the protocol. It may, however, be more efficient to design two paths within each protocol: (1) a "fast" path if the chip has constructed and placed a final group of packets, and (2) a "slow" path if the chip was unable to match the packet. We plan to examine the effects of designing protocols to support two paths in terms of design difficulty and eventual protocol performance. Additionally, because protocols now have access to scattered, demultiplexed groups of packets, it is possible that collapsing various layers of protocols will be efficacious.

The *x*-chip can provide support for distinguishing different quality of services for different packet types. In the most basic sense, the packets that are handled by the x-chip set are receiving a higher quality of support than those that are not handled by the chip. We need to consider protocols that use quality of service as a part of the protocol and determine to what extent effective use of the *x*-chip can improve or simplify these protocols.

It is clear that separating user data and header data is a useful function. How useful is further separation of the user data? In the *x*-chip, we will provide support for some limited user scattering. Will this wider scatter be useful? If the user is allowed to program scatter will there be additional benefits?

What support could the chip provide to gateways? Gateways are generally concerned with routing decisions made on a few bytes. Can additional quality of service support be provided via a fast means of routing packets to different queues? We suspect that pure equality would not be sufficient in the matcher. What additions would the matcher require?

We have chosen to implement two different forms of support for interrupts. We plan to verify that these are sufficient for a large number of existing network protocols, and to investigate whether new protocols can be written to take advantage of this hardware support.

To perform many of these experiments, we need to integrate the software and hardware portions of the protocol in order to determine the effectiveness of the combined system. There are two alternatives for experimentation with the combined system. One is to build a board for the *x*-chip and experiment with it in an actual network. The alternative is to use simulation to investigate its performance. Initially we plan to use the second alternative, which will allow us to begin implementing the simulation system while the *x*-chip is being fabricated. We plan to use an architectural simulator such as Tango[9] or CARL[10] for simulating the system and tcplib[8] to simulate a network load.

4.0.0.2 Prototype Evaluation.

The proposed design of the *x*-chip allows matcher chips to be placed in arrays. The vertical dimension allows for support of more packet types than can be contained in a single chip set, and the horizontal dimension allows for longer matching sequences for each packet type. In addition, the routing chip is designed to support input from these arrays. If the functionality of the *x*-chip is to be embedded in a general network protocol chip, it is likely that a specific choice will be made concerning (1) the types and number of matching units, (2) their lengths, (3) the amount of scatter information supported, and (4) the types of interrupts that will be supported. We intend to identify reasonable and cost-effective choices for these parameters.

In addition, we need to evaluate the cost of shared fields in the matcher. The additional cost required to implement shared fields may not be cost effective. If many of the shared fields are constant, then it may be that these should be hard-wired into the production chip.

There are also issues to be resolved in the router. First, evaluation of scatter width will impact how many scatter locations must be maintained for a production quality chip. In addition, we must evaluate the two types of interrupt support provided by the chip to determine whether they are sufficient, or whether other forms of support are desirable.

Finally, there is some hardware overhead in supporting programming of the chips and their operation concurrently. It is important to determine whether this is cost effective.

5 Status

At this time, the functional design of the *x*-chip is being completed. Detailed design is scheduled to begin in the spring, with fabrication of the first chip scheduled for the end of summer. We plan to use MOSIS, a silicon brokerage service funded by DARPA and NSF, to fabricate the chips. Our CAD tools consist of the VLSI design tools from the University of Washington, augmented by other publicly available software.

References

[1] E. A. Arnould, F. J. Bitz, E. C. Cooper,

[2] H. Kanakia, and D. R. Cheriton, The VMP Network Adapter Board(NAB): High-Performance Network Communication for Multiprocessors. *Proceedings SIGCOMM 88ConferenceonCommunicationsArchitecturesandProtocols'*, 1988.

[3] B. S. Davie, A Host-Network Interface Architecture for {ATM}. *Proceedings SIGCOMM 91ConferenceonCommunicationsArchitecturesandProtocols'*, 1991.

[4] N. C. Hutchinson, and L. L. Peterson, The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, Vol. 17, No. 1, Jan. 1991, pp. 64-76.

[5] J. C. Mogul, Network Locality at the Scale of Processes. *Proceedings SIGCOMM 91ConferenceonCommunicationsArchitecturesandProtocols'*, 1991.

[6] D. L. Tennenhouse, Layered Multiplexing Considered Harmful. *Protocols for High-Speed Networks*. , H. Sudin, and R. Williamson, (Ed.)., Elsevier Science Publishers B.V. (North-Holland), (1989).

[7] D. C. Feldmeier, Multiplexing Issues in Communication System Design. *Proceedings SIGCOMM 90ConferenceonCommunicationsArchitecturesandProtocols'*, 1990.

[8] R. Caceres, P. B. Danzig, S. Jamin, and D.

[9] S. Goldschmidt, and H. Davis, *Tango introduction and tutorial*. Computer Systems Laboratory, Stanford University, 1991.

[10] C. J. Beckmann, *CARL: An architecture simulation language*. CSRD Rpt. No. 1066, Center for Supercomputing Research and Development, University of Illinois, 1990.