A Host Interface Architecture for High-Speed Networks

Peter A. Steenkiste[a], Brian D. Zill[a], H.T. Kung[a], Steven J. Schlick[a], Jim Hughes[b], Bob Kowalski[b], and John Mullaney[b]

[a] School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA

[b] Network Systems Corporation, 7600 Boone Avenue North, Brooklyn Park, MN 55428, USA

Abstract

This paper describes a new host interface architecture for high-speed networks operating at 800 of Mbit/second or higher rates. The architecture is targeted to achieve several 100s of Mbit/second *application-to-application* performance for a wide range of host architectures. The architecture achieves the goal by providing a streamlined execution environment for the entire path between host application and network interface. In particular, a "Communication AcceleratorBlock°(CAB)isusedtominimizedatacopies,reducehostinterrupts,supportDMA and hardware checksumming, and control network access.

This host architecture is applicable to a large class of hosts with high-speed I/O busses. Two implementations for the 800 Mbit/second HIPPI network are under development. One is for a distributed-memorysupercomputer(iWarp)and theotherisforahigh-performanceworkstation (DECstation 5000). We describe and justify both implementations.

Keyword Codes: B.4.1; C.2.1
Keywords: Data Communications Devices; Network Architecture and Design

1 Introduction

Recent advances in network technology have made it feasible to build high-speed networks using links operating at 100s of Mbit/second or higher rates. HIPPI networks based on the ANSI High-Performance Parallel Interface (HIPPI) protocol [1] are an example. HIPPI supports a data rate of 800 Mbit/second or 1.6 Gbit/second and almost all commercially available supercomputers have a HIPPI interface. As a result, HIPPI networks have become popular in supercomputing centers. In addition to HIPPI, there are a number of high-speed network standards in various stages of development by standards bodies. These include ATM (Asynchronous Transfer Mode) [2] and Fibre Channel [3].

Asnetworkspeedsincrease,it isimportantthathost interfacespeeds increaseproportionally, so that applications can bene®t from the increased network performance. Several recent developments should simplify the task of building host interfaces that can operate at high rates. First, most computer systems, including many workstations, have I/O busses with raw hardware capacity of 100 MByte/second or more. Second, existing transport protocols,

in particular Transmission Control Protocol/Internet Protocol (TCP/IP), can be implemented ef®ciently [4, 5]. Finally, special-purpose high-speed circuits, such as the AMCC HIPPI chip set, can be used to handle low-level, time-critical network interface operations.

However, these elements do not automatically translate into good network performance for applications. The problem is that the host interface involves several interacting functions such as data movement, protocol processing and the operating system, and it is necessary to take a global, end-to-end view in the design of the network interface to achieve good throughput and latency. Optimizing individual functions is not suf®cient.

We have designed a host-network interface architecture optimized to achieve high application-to-application throughput. Our interface architecture is based on a *Communication Accelerator Block (CAB)* that provides support for key communication operations. The CAB is a network interface *architecture* that can be used for a wide range of hosts, as opposed to an *implementation* for a speci®c host. Two CAB implementations for HIPPI networks are under development. One is for the iWarp parallel machine [6] and the other one is for the DEC workstation using the TURBOchannel bus [7]. These two CAB implementations should allow applications to use a high percentage of the 100 MByte/second available on HIPPI. The interfaces will be used in the context of the Gigabit Nectar testbed at Carnegie Mellon University [8]. The goal of the testbed is to distribute large scienti®c applications across a number of computers connected by a high-speed network. The network traf®c will consist of both small control messages for which latency is important, and large data transfers, for which throughput is critical.

In the remainder of the paper we ®rst discuss the requirements for the host interface (Section 2). We then present the motivation and hardware and software architecture of the CAB-based interface (Section 3) and the design decisions for the two CAB implementations (Sections 4 and 5). We conclude with a comparison with earlier work.

2 Requirements for a Network Interface Design

In local area networks, throughput and latency is typically limited by overhead on the sending and receiving systems, i.e. it is limited by CPU or memory bandwidth resource constraints on the hosts. This means that the ef®ciency of the host-network interface plays a central role. Consuming fewer CPU and bus cycles will not only make it possible to communicate at higher rates since the communication bottleneck has been reduced, but for the same communication load more cycles will be available for the application. Ef®ciency is critical both for applications whose only task is communication (e.g. ftp) and for applications that are communication intensive, but for which communication is not the main task.

Since host architecture has a big impact on communication performance, we considered the communication bottlenecks for different classes of computer systems. A ®rst class consists of workstations, currently characterized by one or a few CPUs, and a memory bandwidth of a few 100 MByte/second. Existing network interfaces typically allow these workstations to achieve a throughput of a few MByte/second, without leaving any cycles to the application. Some projects have been successful at improving throughput over FDDI, but these efforts concentrate on achieving up to 100 MBit/second for workstations only [9]. This communication performance is not adequate for many applications [10].

General-purpose supercomputers such as Cray also have a small number of processors accessing a shared memory. They have however a very high memory and computing bandwidth and they have I/O subsystems to manage I/O devices with minimal involvement from the CPU. These resources allow them to communicate at near gigabit rates while using only a fraction of

their computing resources [5].

Special-purpose supercomputers such as iWarp [6] and the Connection Machine [11] have a very different architecture. Although these systems have a lot of computing power, the computingcycles are spread out over a large number of relatively slow processors, and they are not suited to support communication over general-purpose networks. A single cell (for iWarp), or a front-end (for CM) can do the protocol processing, but the resulting network performance will match the speed of a single processor, and will not be sufficient for the entire system. The issue is the efficiency of the network interface: can we optimize the interface so that a single processor can manage the network communication for a parallel machine?

Our goal is to define a "Communication Acceleration Block" that can support efficient communication on a variety of architectures. Specifically, this architecture must have the following properties:

1. *High-throughput, while leaving sufficient computing resources to the application.* The goal is to demonstrate that applications on high-performance workstations can achieve several 100 Mbits/second end-to-end bandwidth. It is not acceptable to devote most of the CPU and memory resources of a host to network related activities, so the network interface should use the resources of the host as efficiently as possible, and brute-force solutionsthatmightworkforsupercomputersshouldbeavoided. Theexactperformance will depend on the capabilities of the host.

2. *Modular architecture.* The portions of the architecture that depend on specific host busses (such as TURBOchannel) and network interfaces (such as HIPPI), should be contained in separate modules. By replacing these modules, other hosts and networks can be supported. For example, by using different host interfacing modules, the CAB architecture can interface with the TURBOchannel or to an iWarp parallel machine.

3. *Inherently low-cost architecture.* The host interface should cost only a small fraction of the host itself. It is essential that eventually the host interface can be cheaply implemented using ASICs, similar to existing Ethernet or FDDI controller chips. Early implementations may be more expensive, but the interface architecture should be amenable to low-cost ASIC implementation.

4. *Use of standards.* We concentrate on the implementation of the TCP and UDP internet protocols since they are widely used and have been shown to work at high transfer rates [5]. We use UNIX sockets as the primary communication interface for portability reasons. Wealso wanttobetterunderstandhowprotocolandinterfacefeaturesin uence the performance and complexity of the host-network interface, and other interfaces that are more appropriate for network-based multicomputer applications will be developed in parallel or on top of sockets.

3 The Host-Network Interface Architecture

Many papershave beenpublished thatreportmeasurementsoftheoverheadsassociated with communicatingovernetworks[12,4,13,14,15,16]. Eventhoughitisdifficulttocomparethese resultsbecausethemeasurementsaremadefordifferentarchitectures,protocols,communication interfaces, and benchmarks, there is a common pattern: there is no single source of overhead. The time spent on sending and receiving data is distributed over several operations such as

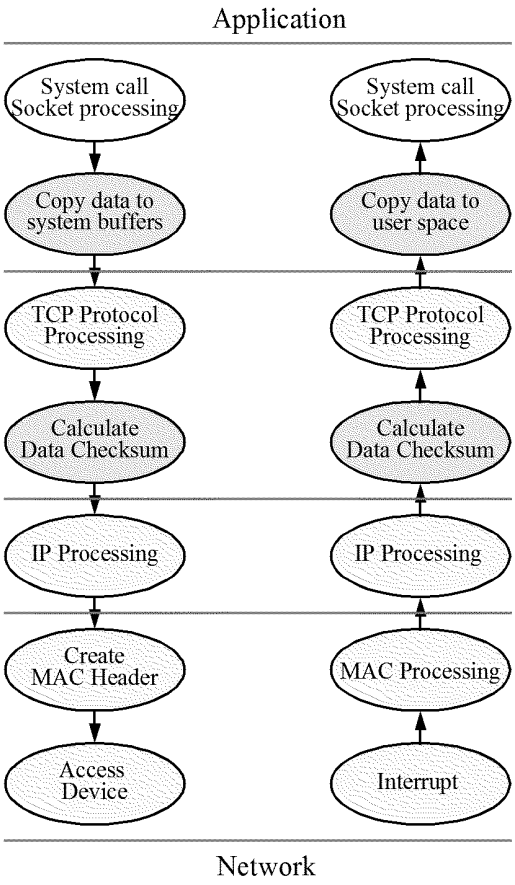

Figure 1: Network data processing overheads

copyingdata, buffermanagement,protocolprocessing, interrupthandlingand system calls, and differentoverheadsdominatedependingonthecircumstances(e.g.packetsize). Theconclusion is that implementing an ef®cient network interface involves looking at all the functions in the network interface, and not just a single function such as, for example, protocol processing.

Figure 1 shows the operations involved in sending and receiving data over a network using the socket interface. These operations fall in different categories. First, there are overheads associated with every application write (socket call ± white), and with every packet sent over the network (TCP, IP, physical layer protocol processing and interrupt handling ± light grey); these operations involve mainly CPU processing. There is also overhead that scales with the number of bytessent (copying and checksumming ± dark grey);this overhead is largely limited by memory bandwidth. In the remainder of this section we ®rst look at how we can minimize both types of overhead. We then present the CAB architecture, and we describe how the CAB is seen and used by the host.

3.1 Optimizing per-byte operations

As networks get faster, data copying and checksumming will become the dominating overheads, both because the other overheads are amortized over larger packets and because these operations make heavy use of a critical resource: the memory bus. Figure 2 shows the data¯ow when sending a message using a traditional host interface; receives follow the inverse

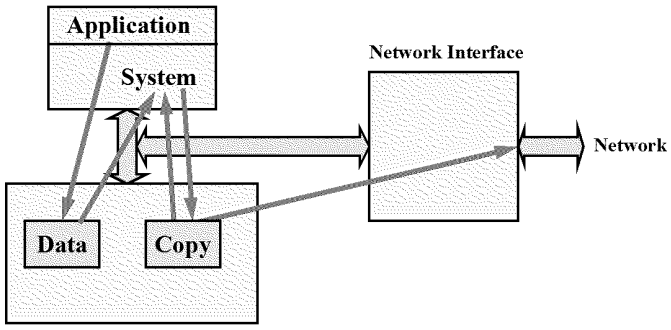

Figure 2: Data¯ow in traditional network interface

path. The dashed line is the checksum calculation. There are a total of ®ve bus transfers for every word sent. On some hosts there is an additional CPU copy to move the data between ªsystem buffersº and ªdevice buffersº, which results in two more bus transfers.

We can reduce the number of bus transfers by moving the system buffers that are used to buffer the data outboard, as is shown in Figure 3. The checksum is calculated while the data is copied. The number of data transfers has been reduced to three. This interface corresponds to the ªWITLESSº interface proposed by Van Jacobson [17]. Besides using the bus more ef®ciently, outboard buffering also allows packets to be sent over the network at the full media rate, independent of the speed of the internal host bus.

Figure 4 shows how the number of data transfers can be further reduced by using DMA for the data transfer between main memory and the buffers on the CAB. This is the minimum number with the socket interface. Checksumming is still done while copying the data, i.e. checksumming is done in hardware. Besides reducing the load on the bus, DMA has the advantage that it allows the use of burst transfers. This is necessary to get good throughput on today's high-speed I/O busses. For example, the DEC TURBOchannel throughput is about 11.1 MByte/second for single word transfers, but 76.0 MByte/second for 32 word transfers. However, on some systems, DMA addsenough overhead that it is sometimesmore attractive to copy and checksum the data using the CPU (see Section 5).

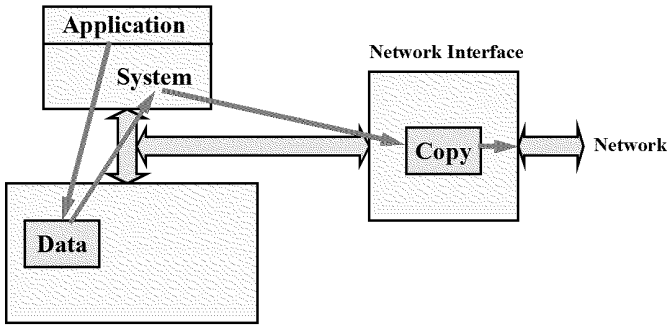

Figure 3: Data¯ow in network interface with outboard buffering

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.