Internet Protocols for Network-Attached Peripherals

Steve Hotz, Rodney Van Meter, and Gregory Finn
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
{hotz,rdv,finn}@isi.edu
tel +1-310-822-1511
fax +1-310-823-6714

Abstract

This paper presents our thesis that the advantages of the internet protocol framework make it the best choice for communications protocols to, and between, network-attached peripherals (NAPs). Moreover, the IP suite is more appropriate than the specialized protocol stacks being developed for commercial NAPs. The benefits of using IP include support for heterogeneous network media, wide-area connectivity, and reduced research and development effort. We examine the issues for use of the internet protocols (TCP/UDP/IP) for NAPs, address commonly held concerns regarding its performance, and describe the Netstation project's prototype implementations of IP peripherals.

1 Introduction

Netstation is a research system architecture based on a switched gigabit network as a system backplane [10]; all of the major subsystems are network attached peripherals (NAPs). A network-attached display has been prototyped, and a software-emulated network-attached disk has been implemented. A camera NAP is also under development, based on the same motherboard as the display.

One of the primary decisions in designing such a system is the choice of a communication infrastructure. To provide maximal ubiquity and independence from specific media technologies, Netstation component communication is based on the TCP/IP protocol suite. This design choice is controversial, in that most commercial NAP efforts (e.g., Tandem's ServerNet [15], Fibre Channel and HiPPI disks and disk arrays) and some research projects (e.g., Minnesota's GFS [21]) use special purpose protocol stacks optimized for performance on specific media. Among research systems, the most similar effort is the High Performance Storage System NAP at Lawrence Livermore [26]. Digital's Petal [18] uses UDP/IP over ATM as part of a large distributed virtual disk system implemented in general-purpose hosts. CMU's Network Attached Secure Disk [13] does use IP for its higher level protocols which are derived from NFS, however, they provide a network "object store" rather than a block-level NAP. The

two projects most closely related to Netstation are MIT's ViewStation [1] and Cambridge's Desk Area Network [14], both of which use ATM to interconnect multimedia peripherals.

The main reason cited for choosing a specialized protocol rather than the media-independent TCP/IP approach is the latter's perceived lack of performance, or the expense to achieve the required performance. We believe that the advantages of TCP/IP merit further consideration for its use in NAP applications, and that the performance concerns are based on inaccurate comparisons with specialized protocol performance. We argue this by examining protocol functionality and its overhead, and by presenting results of our implementation experience.

This paper presents the case for TCP/IP based NAPs. Section 2 discusses the current trend for NAP protocols, provides an overview of the IP protocol suite, and discusses the the use of TCP/IP for NAPs. Section 3 addresses the functionality and potential performance problems of using TCP/IP for NAPs, and Section 4 discusses implementation issues that are often perceived as protocol performance limitations. Section 5 describes the Netstation prototype implementations and their use of TCP/IP, and our conclusions are summarized in Section 6.

2 Protocols for Network Attached Peripherals

Networks such as HiPPI, SSA and Fibre Channel are becoming the access technology of choice for peripherals such as disk drives, tape drives and disk arrays. These networks scale better than traditional I/O channels, connecting more devices over greater distances and providing greater aggregate bandwidth. However, these networks require more complex protocols than are required for traditional bus-based channels such as SCSI.

The NAP community, in most cases, has chosen to use specially developed protocols similar to channel access protocols rather than existing network standards such as TCP/IP, because of perceived differences in functionality, focus, complexity and especially performance. We reason that most of these concerns either reflect misunderstanding of the IP suite or are being met as the suite evolves.

We further argue that the benefits of using IP, including wide-area connectivity, cross-media bridging and reduced research and development efforts, are substantial. Specialized protocols simply do not address inter-LAN communication, which will be important as machine rooms integrate new LAN technologies into increasingly heterogeneous computing environments, and as new uses for sharing NAPs over a wider campus area emerge.

Therefore, we believe that IP is the best choice for storage device peripherals, and should be the protocol selected by NAP system architects.

2.1 IP Framework Overview and Definitions

Throughout this paper we use the terms "IP protocol suite", "IP protocol framework", or "TCP/IP protocol suite". We use these terms interchangeably to refer to a set of protocols that provide the functionality of the network layer and transport layer of

the OSI seven-layer reference model. Specifically, this set of protocols are the DARPA Internet Protocols, and they include the Internet Protocol (IP) at the network layer, and both the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) at the transport layer. It is important to note that there is exactly one network layer protocol, while there are multiple transport layer protocols. UDP and TCP are the most common and thus will be discussed in more detail. However, the framework does allow for other transport layer protocols.

The network layer protocol, IP, provides the basic functionality of exchanging packets of data between any two entities connected to an internet, i.e., a network of networks which may include many different physical media and link layer protocols. The primary philosophy is that this protocol must be ubiquitous: IP runs on everything, and everything runs on IP. IP is intentionally simple and provides very basic functionality which allows it to be implemented on even the simplest of lower-layer protocols. It provides no guarantees on data delivery other than if a packet is delivered, it is delivered to the end host with the specified address. To this end, IP provides a common addressing scheme, a simple header checksum to guarantee that the addresses are not corrupted, and a mechanism to fragment and reassemble packets to accommodate transmission over link layers with different maximum packet sizes.

The transport protocols, TCP and UDP, build on top of IP, exchanging their own headers and data payloads using the basic facilities provided by IP. These protocols use different mechanisms to provide various levels of end-to-end functionality. UDP provides simple user-level access to the basic unreliable service provided by IP, and as such, it adds very little overhead to the exchange of data. On the other hand, TCP provides for a reliable, in order, byte stream and is by far the most complex of the three protocols discussed here.

There exists other related and supporting protocols, such as an Address Resolution Protocol (ARP) for mapping IP addresses to link-layer addresses, and various routing protocols to exchange information so that intermediate internetwork nodes can determine the correct disposition and forwarding of IP packets to their destination. However, these other protocols do not directly impact the per-packet data processing overhead at end systems, and consequently are only mentioned briefly in this paper.

2.2 IP for NAPs

Choosing IP as the communication infrastructure for NAPs alleviates the problems of committing to a protocol suite which is tied, for the most part, to the choice of physical media, e.g., Fibre Channel or ATM. This provides a growth path that is unconstrained by the future development of a particular technology. In the same vein, IP allows for cross-media bridging with minimal incremental effort. Cross-media bridging will be useful given increasingly heterogeneous computing environments, allowing transparent interoperation among different types of networks. Finally, the wide area connectivity that is IP's strength opens up new functionality for peripherals, for example, remote mirroring of disk drives and remote backup.

Using IP will allow NAP developers to exploit the large existing body of research and development in flow control, routing, congestion control, and reliability. This will reduce R&D effort, as well as allowing quick integration of emerging features such as resource reservation and real-time protocols. The effort to adapt and optimize IP implementations for the NAP environment should prove less costly than reinventing pieces of the network solution as new NAP requirements are realized.

We believe the concerns about IP performance are founded on comparisons between IP and specialized protocols, and that the performance differences are more dependent on implementation and environment than on the complexity of the protocols themselves. IP performance measurements are typically made in general purpose (and often outdated) operating systems with protocol implementations tuned for wide-area communication using relatively small packets, while NAP protocol performance benefits from a low-cost embedded operating environment, large data packets, and specialized protocol coprocessors.

These concerns about IP performance and complexity should prove to be non-issues. The issues and differences can be classified and addressed in a general way as follows:

- **Protocol functionality required by NAPs, already provided by IP.** NAPs will benefit from the functionality such as segment size negotiation, reliability, flow control, and cross-media bridging already provided by IP. These common functionality requirements are the main motivation driving the use of IP for NAPs.

- **IP protocol functionality and complexity not required by NAPs.** Concerns about the performance impact of additional "baggage" functionality can be addressed by fast-path and common-case implementations. As evidence, we note that the combined transport (TCP) and network (IP) processing of common-case packets has already been optimized to approximately 200 machine instructions, which would cost 2-8 microseconds on a typical NAP-embedded microprocessor.

- **Protocol functionality required by NAPs, not provided by IP.** While NAPs can take advantage of IP to avoid re-inventing a variety of technologies and functionality, there are clearly cases where the NAP community can (and must) extend the IP framework. Support for application-layer framing and fast application demultiplexing might be addressed by transport-layer options, or may require a new transport protocol to complement the existing UDP and TCP. Section 3.3 discusses these issues in more detail.

- **Implementation and operating system issues.** Many of TCP/IP's known performance problems come from outdated implementations or restrictions imposed from outside, such as data copies through the UNIX socket API and ethernet's small packet size limit. This overhead is considerable compared to the actual fast-path protocol processing. These problems will disappear as the protocol stack is moved into the highly-optimized embedded NAP environment. Section 4 discusses these issues in more detail.

We reason that as systems move to larger, more complex switched networks for I/O, some sacrifice of performance is the inevitable result. However, IP has no inherent performance penalty relative to other protocol choices; the loss is entirely attributable to managing the additional complexity. In this environment, IP offers significant advantages and few drawbacks. IP, therefore, should be the network protocol of choice for network-attached peripherals.

2.3 NAP Implementations of the IP Protocol Suite

We do not expect that performance-critical NAPs will run based on current IP implementations, or without additional development of the IP protocol suite. Rather, there is an open IP framework for network communication, and NAP networking development could benefit from working within this framework. Within this framework there are both design changes that can not be made, and considerable flexibility to accommodate NAP requirements.

The discussion in Section 2.1 points out those parts of the IP suite that will not change, specifically, the IP network layer protocol itself. To be an IP NAP, a device must implement the IP protocol.

There are three general approaches that NAP developers can take to adapt the IP protocol suite for their devices:

- **Use protocol options.** Protocols of the IP suite, including IP and TCP, define protocol options that allow the protocol to provide additional required features on a per-use basis, e.g. per IP packet or per TCP connection. If existing options do not provide for efficient NAP communications, new options can be introduced within the IP protocol framework.

- **Optimize implementations for expected NAP usage patterns.** IP protocol implementations can benefit from many of the same optimizations that facilitate specialized NAP protocol performance. Section 4 discusses these issues in more detail.

- **Develop an alternative IP-based transport protocol.** TCP and UDP are the most widely used transport protocols in the IP suite, however, alternate transport protocols can also be supported within this framework (section 2.1). Sections 3.2 and 3.3 discuss transport protocol issues in more detail.

We believe these methods will allow NAP developers to adapt the IP protocol suite to their high-performance requirements, and gain the benefits of IP's inter-LAN connectivity. Further, the development effort to adopt the IP suite can build on IP's years of development experience to solve NAP-specific issues, and this effort should be less complex than the effort to develop media-specific protocols for NAPs.

3 TCP/IP Protocol Stack Issues

This section examines the functionality of the TCP/IP protocol to address the issue of complexity, and determine why the performance may be suitable (or unsuitable) for NAPs.

Our general premise is "you get what you pay for"; functionality required by NAPs will have approximately the same overhead, regardless of whether it is implemented within an IP framework or in a specialized NAP protocol stack. Further, the overhead for portions of IP not required by NAPs can be minimized by fast-path, common-case implementations. The transport (TCP) and network (IP) processing of common-case incoming packets has already been optimized to approximately 200 machine instructions. This number was observed in [6], and is confirmed by analysis of our TCP/IP implementation for the Netstation display. This would allow a baseline 250,000 packets/sec on a relatively humble 50MIPS embedded NAP processor. This simple analysis indicates that the TCP/IP processing is not a performance bottleneck that would make it unsuitable for NAPs.

3.1 Network Layer Processing

Examination of the IP network-layer implementation shows that the IP end-host processing requires a relatively small amount of code. There are four primary functions that would affect an IP NAP: packet header processing, header checksum calculation, byte-swapping, and fragmentation reassembly. None of these functions represent a prohibitive processing cost.

Processing of the 20-byte IP header is straightforward, and should not represent a significant incremental overhead beyond processing for a specialized NAP protocol. The required processing on outgoing packets generally involves an efficient bulk copy of a 5-word template, and then 4 load and store operations to modify the packet identifier, length, destination address, and checksum fields. To verify proper reception, a simple implementation would have a compare and branch operation for each of the 12 header fields, although optimization can reduce this even further.

The short IP header checksum calculation involves 12 16-bit add operations, 2 shifts and 3 mask operations; the nature of the checksum would allow optimizations using larger (32-bit) add operations. Although the header checksum itself is not a prohibitive cost, data checksumming incurs considerable overhead; we address data checksumming in Section 3.2.

Byte-swapping is necessary only for little-endian machines, where byte order does not match network standard byte order [7]. However, it is an operation that may affect all network communications, thus this discussion applies to transport layer processing as well. Moreover, this implies that even in a non-IP NAP, this function is required to support heterogeneous hosts.

The most complex, and potentially expensive, functionality we must deal with is IP fragmentation reassembly. Fortunately, we can virtually eliminate this processing by sending appropriately sized transport layer segments. Sending such segments should be straightforward for common-case NAP applications on a single local area network (LAN) where the maximum frame size is known. Mechanisms such as MTU discovery [19] can provide the analogous information across a WAN. For both cases, higher layer communication protocols can avoid packet fragmentation by negotiating the segment size; this is specifically supported by TCP's Maximum Segment Size option.

3.2 Transport Layer

Transport layer functionality is more complex than that of the network layer. In the case of TCP, it provides for in-order reliable delivery, with a number of window and congestion control mechanisms. However, we also do not believe transport layer functionality will prove prohibitive, rather this is the area where NAP developers would focus their efforts.

The general reasons that we believe transport functionality in an IP stack is not prohibitive are as follows:

- **UDP is available as a lower-overhead alternative.** Assuming that NAPs will be commonly accessed within a local environment, the general-purpose functionality provided by TCP may not be needed. UDP provides a very simple protocol that provides for connection demultiplexing and checksumming. Both our Netstation VISA protocol (Section 5.2) and the common NFS protocol [22] implement their own simple reliability for LAN communications on top of UDP.

- **Fast-path TCP is already reasonably efficient.** As noted earlier in the paper, the common-case TCP overhead allows for many thousands of packets per second, provided system issues such as data copies, context switching, and checksumming are addressed. These issues must be dealt with in either the case of an IP or a non-IP NAP.

- **IP can support other transport protocols.** This paper argues primarily for an IP NAP, and that there would be considerable advantage if NAP development could exploit either of the existing transport protocols (UDP and TCP) by providing implementations optimized for the NAP environment. However, IP already supports multiple transport protocols, and recasting current NAP-specific transport protocols into the IP framework seems a viable alternative. New general-purpose transport protocols face a practical hurdle on account of the large installed base of TCP and UDP, and need for widespread deployment and backward compatibility. However, a NAP transport protocol could be deployed incrementally as NAPs are integrated into the computing environment, which would mitigate this issue of ubiquitous deployment.

These general observations allow for the feasibility of an IP NAP with appropriate transport functionality. Due to the number of transport protocols and different levels of service, we cannot address the spectrum of transport protocol functionality. However, the most important transport functionality that a NAP would have to address will be common to any protocol, specifically data checksumming and retransmission and reliability. The issues of connection demultiplexing and segmentation and reassembly are addressed in the following section.

Computing a checksum over the entire data packet is one of the biggest obstacles to low-overhead transport performance. In the case of a NAP (or any network node), either reliability guarantees are required from the transport layer or they are not. If this functionality is to be provided, the checksum overhead is incurred regardless of the

particular protocol framework. If a checksum is not required or is not used by a NAP-specific protocol, then an IP-based stack can also eliminate this overhead. Currently, the UDP specification allows for non-checksummed payloads, and in the future, NAP developers could incorporate this functionality into their preferred transport layer protocol. Efficient checksum implementation is discussed in Section 4.

Our reasoning about reassembly and reliability mechanisms at the transport layer is much the same as for checksumming; either this functionality is needed, or the media layer provides sufficient guarantees to allow a simpler mechanism. In the former case, TCP has years of development lead time, in the latter case UDP provides a simple low-cost framework, and there is the option of developing another transport protocol to run over IP.

3.3 Open Issues

We classify the efforts required to achieve IP NAPs as either addressing implementation and system issues (discussed in Section 4), or as protocol development efforts. The latter includes (1) the selection, design, and development of a transport protocol and transport protocol options, and (2) providing support for application layer functionality.

The transport layer issue is of practical as well as technical importance. The perceived lack of a suitable standard transport protocol is perhaps the most significant barrier to the development of IP NAPs. We have also run into this issue within the Netstation project, and are not 100% decided after having experimented with UDP, TCP, and our own specialized transport protocol designed to run over IP. The main action item is a study to decide whether (1) an enhanced implementation and operating system for TCP will provide sufficient performance, (2) a standardized simple reliability mechanism can be effectively provided on UDP, or (3) whether one of the current media-specific protocols should be brought into the IP framework.

Support for high-performance application layer functions is also an open issue. In particular, application layer framing and fast application demultiplexing may be critical functionality. Currently, support for these issues is minimal.

Application layer framing is concerned with providing the efficient communication of application-sized data chunks (e.g. a disk block or track) using packet sizes provided by the lower layers. A simple case of this would be using a path MTU (maximum transmission unit) discovery mechanism to find that the physical network supports 20Kbyte packets, using TCP's Maximum Segment Size option to coordinate the transport layers, and providing this information to the application so it can send an integral number of its data blocks. However, this issue becomes more complex as application data units become larger than a single network packet. Application layer framing not only deals with protocol issues such as packet sizes and transport-layer fragmentation, but also must be integrated with other sources of overhead in the system such as interrupts and context switches (per network packet or per application data unit) and data copies often required for packet reassembly.

Fast application-level demultiplexing and delivery is another issue, and one of the areas media-specific protocol stacks are addressing (e.g., Fibre Channel allows data

blocks to be passed directly into pre-defined memory locations at a receiver, achieved by processing all network protocols in the host adapter). The first component of this issue is simply an efficient mechanism to associate an incoming packet with the expecting receiver. If there are thousands of open connections, a poor implementation of TCP, for example, can require considerable overhead to find the appropriate match on the tuple:

<source address, destination address, source port, destination port>

Implementations can use hashing or caching of recently active session to improve performance, but another option to examine is providing a short header "handle" that applications may pass back and forth. Providing for application-specific delivery or disposition of data is also an important area of research [9, 16, 2, 27].

4 Implementation and System Issues

Many of TCP/IP's known performance problems come from outdated implementations or restrictions imposed from outside. In this section, we address some of those concerns in the context of implementing TCP/IP inside a network attached peripheral.

Changing to larger data units can dramatically reduce the CPU load at high data rates. On media that support large packet sizes, the first step in reducing CPU utilization and increasing throughput is to increase the packet size, which IP supports. For disk drives, data payloads which conveniently map to file system pages, such as 4KB, may be particularly efficient.

Much of the CPU cost of networking in general-purpose hosts comes from context switches and virtual memory management. With a lightweight, real memory embedded operating system in the NAPs, context switches are inexpensive. Without virtual memory, maintenance of page tables, with their associated locking, mapping and protection, is unnecessary.

Reduction of data-touching operations is required to achieve high performance with minimal CPU cost. Data touching takes two forms, data copies and checksum calculation.

Data copies are reduced through integrated layer processing and possibly the use of zero-copy APIs, without the overhead of the Berkeley sockets layer. Data may be shared directly between the "application" (the SCSI command processor/disk drive track buffer manager, in the case of a disk drive) and the networking stack [8]. Fast demultiplexing of received data can also be used.

The overhead of the checksum can be eliminated in several ways. The simplest but least desirable option is to eliminate it and depend on the LAN checksum to protect the data; this is common today for UDP, but does not protect data end-to-end or across network boundaries. Alternatively, the CPU can calculate the checksum in conjunction with a data copy [20]. This can generally be done for zero cost on RISC processors by putting the adds in the delay slots of loads and stores. The checksum can also be calculated on any data movement, such as the reception from the network or from the disk platter into buffer memory, with simple hardware modifications [11, 23].

It is possible to store the TCP checksum on disk with the data by reformatting the disk with a larger sector size. Because the checksum is additive, storing it with each sector allows quick calculation regardless of the packet boundaries used for transmission.

The remaining per-packet CPU costs can be controlled by careful use of a "fast path" through the system, in which the common case is optimized at the expense of unusual cases. For example, TCP header prediction assumes that most packets are either an in-order data packet or the next expected ACK, and simple optimized tests for these cases appear very early in the code. This improves performance for the majority of packets, but is additional (and redundant) processing for packets arriving out of order.

5 Netstation Implementation Experience

The Netstation Project has implemented prototype IP NAPs as part of its research into a LAN-based system architecture. We have built a display NAP based on a custom hardware design, and are currently completing a camera NAP based on the same hardware components as the display. Other NAPs, including the IP disk and keyboard, are emulated using SUN workstations. Netstation uses Sparc 20/71 workstations running SunOS 4.1.3 as the "CPU nodes" that control and access the IP NAPs.

The remainder of this section provides a summary of the Netstation Project, and describes our IP NAP implementations with a focus on the IP stack performance and overhead.

5.1 Netstation Overview

Netstation [10, 12] is a heterogeneous distributed system composed of processor nodes and peripherals (NAPs) attached to a high-speed LAN which provides high aggregate bandwidth; currently we use both a 640 Mbps Myrinet network and a 100 Mbps ethernet.

Netstation is predicated on the observation that shared I/O buses provide poor scalability, and are falling behind the technology curve of high-speed LANs. By connecting peripherals directly to a switched high-speed LAN, a Netstation system can avoid the shared-bus bottleneck. This allows Netstation components to communicate directly with each other, without intervention of the main system processor or server processor, e.g. a user can initiate a disk backup to tape, and have negligible impact on a resource-intensive multimedia conference. This sharing of resources also creates flexibility in system configuration.

The ultimate Netstation goal is a ubiquitous network-based architecture, but the results can also be applied individually to produce NAPs. The Netstation project concentrates on operating systems mechanisms and network protocols for NAPs. Netstation provides a single system image, allowing the CPU to access the NAPs in a manner as similar as possible to directly attached devices. Because the devices are attached to an open network with both trusted and untrusted nodes on the net, safe shared access and security at the NAPs are critical.

Netstation has developed an abstraction we refer to as the *derived virtual device*, or DVD [25], that provides the mechanisms for safe shared device access. DVDs provide a protected execution context at the device, allowing direct use of the devices by untrusted clients, such as user applications. The owner of a device, or its managing process, specifies the DVD resources, DVD interface, and the security policy, then initializes this configuration at the NAP; in turn, the NAP's DVD provides the specified interface to these resources and enforces the access policy. We believe that DVDs can accommodate all of the issues regarding shared access to NAPs, including third-party I/O transfers. Thus, a camera can be granted write access to a restricted region of a frame buffer, or a user application can be given read-only access to a DVD which represents a disk-based file or disk partition.

Netstation has a number of IP NAP prototypes in various stages of development. A custom motherboard provides the basis for our Netstation display NAP, which provides a network-attached frame buffer interface. The display is controlled by an adapted version of the MIT X11R5 server, which drives its frame buffer remotely across the network. A camera NAP based on the same motherboard is under development. Support for third-party I/O, in which video will be transferred directly from the camera NAP to the display NAP, is a near-term goal. Netstation has also implemented an *IPdisk*, an emulated network-attached disk drive. Access to the IPdisk is made via the Virtual Internet SCSI Adapter (VISA), which provides a SCSI-to-DVD translation to support access to storage peripherals via the network. A simple keyboard NAP was also prototyped earlier in the project.

Described below is our experience with two different IP stack implementations; a summary of the standard SunOS IP stack performance for the IPdisk, and a more thorough discussion of our own TCP/IP implementation for the custom Netstation display system.

5.2 IPdisk Performance and Issues

IPdisk is our emulated network-attached SCSI disk drive. It currently runs as a user process on a Sun workstation. It provides all the mandatory commands for a SCSI direct access device, including TEST UNIT READY, RESERVE, RELEASE, READ, WRITE, FORMAT, and others. Commands and data can be sent via either TCP connections or UDP datagrams with a simple reliability mechanism. Support for third party copy between IPdisks in cooperation with the file system is currently under development. This will move data directly from disk to disk via the network, without consuming memory or bus bandwidth at the controlling host.

VISA, our Virtual Internet SCSI Adapter, is an operating system module added to SunOS. It implements a `scsi_transport` layer in the layered device driver system, accepting commands from the higher-level SCSI disk and tape drivers and packaging and transmitting them to the appropriate devices. The SCSI disk driver and all of the file system components are completely standard; a normal fast file system is built on top of the IPdisk. The VISA module is roughly 2,000 lines of C code added to the SunOS kernel, less than the amount for the standard SCSI host bus adapter.

Preliminary performance measurements indicate that VISA is capable of running

at a write rate of 72 Mbps (megabits per second) on a 75MHz SPARCstation 20/71 through the file system to IPdisk via UDP over Myrinet. The same CPU is predicted to reach 110 Mbps through the file system over an infinitely fast SCSI bus to an infinitely fast disk, limited primarily by the OS's ability to manage the virtual memory system. Known optimizations and reductions in the number of data copies could be expected to raise the data rate to 90-95 Mbps, or above 80% of that achievable with a SCSI bus, without the addition of special network coprocessors. We feel this performance is acceptable, and that this supports our premise that the advantages of IP outweigh the performance concerns.

A more detailed description of VISA and its performance can be found in [24].

5.3 Netstation Display

The Netstation display hardware is a custom motherboard centered around TI's 40MHz TMS-320c80 MVP chip, selected primarily for its high performance video and data transfer controllers. A PCI bridge chip (V3 Semiconductor's V960PBC) provides a path from the main C80 bus to an off-the-shelf Intel EtherExpress PR0/100 interface card and shared packet buffer memory. The display hardware also includes an audio subsystem and provisions for two JPEG decompression cards that are under development.

The display operating system is based on the software in the TI development kit, specifically the C8x Multitasking Executive, release 2.0, and the accompanying runtime libraries. The C8x executive provides basic synchronization, communication, and task primitives to support cooperative multitasking in a single shared memory space. This executive is fairly lightweight and efficient; the profiled baseline kernel performance takes 115 clock cycles (2.875 microsecond at 40 MHz) to preempt, switch context, and transfer a message between two tasks.

The TCP/IP stack, including device drivers for the PCI bridge and Intel Ethernet board, is built on top of the OS primitives. Initially, we facilitated development by instantiating a simple task to handle each layer and each side (send and receive) of the communication stack. Over time, we have revised the structure of the system to include two primary tasks: (1) a task to handle the low-level driver, and to process the receive-side of ethernet and IP, and (2) a TCP task to provide the application's network API, which also processes IP and ethernet send-side functionality before passing packets to the low-level task for transmission.

Data copies in the system are kept to a minimum by exploiting the shared-memory model of the OS. Currently, a sending application copies data into buffers to be sent, and after being passed between tasks by reference, the data is copied out of the system to the ethernet board for transmission; the receive side incurs two analogous copies. We believe the current structure would allow us to eliminate one copy in each direction with minimal effort, using mechanisms such as scatter-gather vectors and requiring application cooperation to manage buffer memory.

5.3.1 TCP/IP for the Netstation Display

The network and transport layer functionality currently implemented for the Netstation display includes: IP, TCP, Address Resolution Protocol (ARP), and the echo function of the Internet Control Message Protocol (ICMP). We have not yet implemented UDP, however, it is straightforward and would be much simpler than TCP to implement. The ICMP function was included for testing and debugging of the low-level drivers. We believe these five components represent the minimal stack needed to communicate in a TCP/IP environment, and that this is a reasonable amount of functionality to include in a NAP.

The IP, ARP, and ICMP protocols were implemented from scratch. Our TCP implementation is based on INRIA's user-level TCP [5], which in turn is based on BSD TCP. Note that our TCP implementation does not run as part of the user task, rather we take advantage of the low-overhead kernel primitives and manage TCP as a separate task. We also manage data buffers and the user API differently than either the INRIA or Berkeley implementation.

The amount of memory required to implement this functionality at a NAP is also reasonable, and should not be a burden on potentially limited NAP resources. Further, assuming the amount of code is indicative of the effort required for the development of an IP NAP, this too seems reasonable. Table 1 enumerates the amount of C code and the memory footprint of the Netstation networking and transport layer stack, with the OS primitive information provided as a point of comparison.

Component	C-Code (lines *.c)	Memory Footprint (bytes of static code)
ICMP	90	184
ARP	570	2804
IP	1210	6444
TCP	2700	12636
shared code	390	1248
user lib	630	3908
TCP/IP Total	5590	27224
Reference pts:		
OS Primitives	3400	7708
OS Libraries	2300	3860

Table 1: TCP/IP Code Size

Other functionality (e.g. DNS, RARP, SNMP, etc) would be useful to facilitate monitoring, configuration, and other administrative functions, and the existence of these networking standards might further simplify the development of commodity NAPs. However, these additional components are not necessary for NAP communication, and should not be considered part of the "TCP/IP baggage" that might prevent the adoption of IP for NAPs. An actual IP host will have additional functionality, per the Host Requirements RFCs [4, 3], however, we believe IP NAPs would

have a similar, but reduced, set of requirements.

The performance of the display TCP/IP implementation is currently being evaluated, concurrently with final hardware and software debugging and tuning.

6 Summary

This paper suggests that the TCP/UDP/IP framework offers considerable advantages for network-attached peripherals, in particular cross-media bridging, ubiquity, and the existence of considerable prior work. We have argued that using IP for network attached peripherals is preferable to the development of specific networking protocols optimized for each new physical media.

We have addressed the primary concern regarding IP for NAPs, pointing out that the performance of IP is strongly dependent on the operating system environment and the particular implementation. We suggest that IP performance would be sufficient, if the efforts to develop NAP specific protocols were redirected toward optimizing IP for the NAP environment.

Finally, we have described the Netstation project's use of the TCP/IP network protocol suite as the means to access several different types of peripherals, including disk drives and displays. We have shown how access to our IPdisk using VISA can perform at over 80% of a native SCSI disk performance using simple known optimizations to SunOS UDP. There is reason to believe that additional performance improvements would be achieved in a typical light-weight embedded NAP implementation.

Our conclusion is that IP is the appropriate choice for interconnecting subsystems and that the desired performance is feasible, although additional work is required before the performance reaches sufficient levels.

Acknowledgments

This research was sponsored by the Defense Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of DARPA, the U.S. Government, or any person or agency connected with them.

References

[1] J. F. Adam, H. H. Houh, M. Ismert, and D. L. Tennenhouse. Media-intensive data communications in a "desk-area" network. *IEEE Communications*, pages 60–67, Aug. 1994.

[2] M. L. Bailey, M. A. Pagels, and L. L. Peterson. The x-chip: An experiment in hardware demultiplexing. In *Proceedings of the IEEE Workshop on High Performance Communications Subsystems*, Feb. 1991.

[3] R. Braden. Requirements for internet hosts - application and support. Internet Draft RFC 1123, USC/ISI, Oct. 1989.

[4] R. Braden. Requirements for internet hosts - communication layers. Internet Draft RFC 1122, USC/ISI, Oct. 1989.

[5] T. Braun, C. Diot, A. Höglander, and V. Roca. An experimental user level implementation of TCP. Technical Report RR-2650, INRIA, Sept. 1995.

[6] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An analysis of TCP processing overhead. *IEEE Communications*, 27(6):23–29, June 1989.

[7] D. Cohen. On holy wars and a plea for peace. *IEEE Computer*, Oct. 1981.

[8] P. Druschel and L. L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *Proc. Fourteenth ACM Symposium on Operating Systems Principles*. ACM, Dec. 1993.

[9] D. R. Engler and M. F. Kaashoek. DPF: Fast, flexible message demultiplexing using dynamic code generation. In *Proc. SIGCOMM '96*, volume 26, pages 53–59. ACM, Oct. 1996.

[10] G. Finn. An integration of network communication with workstation architecture. *ACM Computer Communication Review*, Oct. 1991. Available online at http://www.isi.edu/netstation/.

[11] G. Finn, S. Hotz, and R. Van Meter. The impact of a zero-scan internet checksumming mechanism. *ACM Computer Communication Review*, 26(5):27–39, Oct. 1996.

[12] G. G. Finn and P. Mockapetris. Netstation architecture: Multi-gigabit workstation network fabric. In *Proc. NetWorld+InterOp Engineer Conference*, 1994.

[13] G. Gibson et al. A case for network-attached secure disks. Technical Report CMU-CS-96-142, CMU, June 1996.

[14] M. Hayter and D. McAuley. The desk area network. *ACM Operating Systems Review*, 25(4):14–21, Oct. 1991.

[15] R. W. Horst and D. Garcia. ServerNet SAN I/O architecture. In R. Rettberg and W. Dally, editors, *Hot Interconnects Symposium V*. IEEE Computer Society, 1997.

[16] J. S. Kay. *Path IDs: A Mechanism for Reducing Network Software Latency*. PhD thesis, UCSD, 1995.

[17] B. Kobler, editor. *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, Sept. 1996.

[18] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proc. ACM Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Oct. 1996.

[19] J. Mogul and S. Deering. Path MTU discovery. RFC 1191, Internet Request For Comments, 1990.

[20] C. Partridge and S. Pink. A faster UDP. *IEEE/ACM Trans. on Networking*, 1(4):429–440, Aug. 1993.

[21] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The global file system. In Kobler [17], pages 319–342.

[22] Sun Microsystems Inc. NFS: Network file system protocol specification. Technical Report RFC 1094, Internet Request For Comments, 1989.

[23] J. Touch and B. Parham. Implementing the internet checksum in hardware. Technical Report Internet RFC 1936, ISI, Apr. 1996.

[24] R. Van Meter, G. Finn, and S. Hotz. VISA: Netstation's virtual internet SCSI adapter. in preparation.

[25] R. Van Meter, S. Hotz, and G. Finn. Derived virtual devices: A secure distributed file system mechanism. In Kobler [17].

[26] D. Wiltzius and K. Minuzzo. Network-attached peripherals (NAP) for HPSS/SIOF. web page, Oct. 1995. http://www.llnl.gov/liv_comp/siof/siof_nap.html.

[27] M. Yuhara, B. N. Bershad, C. Maeda, and J. E. B. Moss. Efficient packet demultiplexing for multiple endpoints and large messages. In *Proc. USENIX Winter 1994 Technical Conference*, Jan. 1994.