IEEE Global Telecommunications Conference

GLOBECOM'91

Phoenix, Arizona • December 2-5, 1991

"COUNTDOWN TO THE NEW MILLENNIUM"

Featuring a Mini-Theme on:
Personal Communications Services (PCS)

Conference Record Vol. 3 of 3

VOLUME	DAY	SESSIONS	PAGES
1	Tuesday	1-20	1-694
2	Wednesday	21-42	695-1531
3	Thursday	43-60B	1533-2150

IEEE

IEEE

ICC GLOBECOM

Sponsored by the
IEEE Communications Society
and the Phoenix IEEE Section

Additional sets of Volumes 1, 2, and 3 may be ordered from:

IEEE Service Center
Publications Sales Department
445 Hoes Lane
P.O. Box 1331
Piscataway, New Jersey 08855-1331

IEEE GLOBECOM '91

IEEE Catalog No.:	91CH2980-1	
ISBN Numbers:	0-87942-697-7	Softbound
	0-87942-698-5	Casebound
	0-87942-699-3	Microfiche
Library of Congress No.:	87-640337	Serial

AN OUTBOARD PROCESSOR FOR HIGH PERFORMANCE IMPLEMENTATION OF TRANSPORT LAYER PROTOCOLS

R. ANDREW MACLEAN and SCOTT E. BARVICK[1]

Bellcore, 444 Hoes Lane, Piscataway, NJ 08854

ABSTRACT: The high throughputs promised by emerging network technologies are often difficult to achieve application-to-application because of host transport protocol bottlenecks. This paper describes an experimental prototype implementation of an outboard protocol processor which eliminates these bottlenecks by performing transport layer functions in dedicated hardware. The architecture consists of separate transmit and receive CPUs, each with checksum and DMA circuits. Measurements made using an implementation of the TCP protocol indicate that this architecture can support end-to-end throughputs in excess of 11,000 packets/sec between UNIX[2] hosts.

1. INTRODUCTION

MEASUREMENTS of the end-to-end performance of LANs indicate that transport and network layer protocol implementations often limit throughput between communicating applications [1]. Several solutions have been proposed, the most common of which can be categorized as follows:

1) increase the size of the transport protocol data unit (TPDU),
2) optimize the protocol software,
3) change to a more efficient protocol and
4) use a more powerful host processor.

The idea behind the first method is to increase the ratio of data bits to header bits so that the protocol processing required per data bit is reduced. Depending upon the underlying network, however, this approach may lead to increased latency in the network, or the need for fragmentation and reassembly in the lower layers.

Optimization of the communications software for throughput has been discussed for the case of TCP/IP [2]. The methodology adopted is to change the existing implementation software to reduce the protocol processing overhead. Performance comparisons of optimized and non-optimized implementations of TCP/IP found in [1] indicate that significant performance increases can be realized using such techniques..

The TCP and OSI TP4 protocols have been designed primarily for robustness and utility rather than throughput. Several 'lightweight' transport layer protocols have been proposed which offer performance improvements: NETBLT [3], XTP [4], NACK [5], SNR[7], and VMTP [6] are examples of such protocols. With respect to the hardware, typically, the data link layer (to use OSI terminology) has been handled by some kind of host network adapter and the network layer and above has been the responsibility of the host processor. Thus there are potentially two areas where processing power can be added, the host or the communications adapter.

While the use of more powerful host processors is a common solution, there is a growing trend towards increasing the front end intelligence of the I/O. There are two primary reasons for this; firstly, the I/O subsystem often demands response times which cannot be guaranteed by the host processor or would cause the host to behave inefficiently or erratically. Secondly, it is often the case that certain compute intensive functions can be performed more quickly or more cost effectively by specialized hardware than by the host processor. Several outboard processor designs have been reported [7]-[12]. Our objective for this project has been to explore the outboard approach by designing an experimental prototype processor as a platform for analyzing transport layer protocols. We call this processor the Protocol Accelerator (PA), and it is described in the sections which follow. One of our ultimate aims is to explore different high speed protocols using this processor as a platform, in order to determine the most appropriate techniques for transport of data on high speed Metropolitan Area Networks.

2. PROTOCOL ACCELERATOR

2.1 System Configuration

The Protocol Accelerator is a board on the VME system bus. Figure 1 shows how the PA integrates into the host system. On the network side, the PA is equipped with both input and output 32 bit parallel ports, each supporting data transfer rates in excess of 320 Mbits/sec. Intentionally, no media access circuitry has been included on the PA; this provides us with the capability to connect the Protocol Accelerator to a variety of network types via appropriate adapters, or, as in the case of loop-back testing, to leave out the network circuitry completely. In future communications subsystems, the transport protocol acceleration circuitry probably would physically reside on the network adapter card.

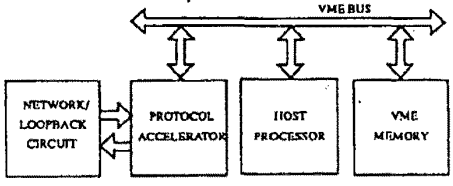

FIGURE 1. System Configuration

1. Scott Barvick is now with Wellfleet Communications, 15 Crosby Drive, Bedford, MA 01730
2. UNIX is a registered trademark of UNIX System Laboratories Inc.

CH2980-1/91/0000-1728 $1.00 © 1991 IEEE

2.2 Functionality and Data Flow

The previously reported outboard processors can be categorized into single and multiple CPU architectures. The single CPU implementations [4, 10, 11] include peripheral support for high data throughputs. The multiprocessor implementations [7, 8, 9, 12] use up to eight CPUs, typically with no special peripheral devices. In our design, we exploit features from both of these approaches with a dual CPU design and special purpose peripheral circuitry in an architecture optimized for transport layer processing.

The internal functions and data flows of the protocol accelerator are shown in Figure 2. We use a dual CPU approach to protocol processing, with one CPU subsystem dedicated to the transmission, and the other to the reception. The transmit and receive CPUs are both 68020 (25 MHz) based, each with its own private resources: ROM, parallel I/O, interrupt circuitry and 128 kilobytes of random access memory (RAM). In addition there is 128 kilobytes of RAM shared by both CPUs which is also accessible to the two host busses, VME and VSB. Using both host busses simultaneously, it is possible for the PA to move data blocks both to and from host memory. The transmit and receive CPUs have VME bus master capability. All the data paths shown are 32 bits wide.

2.3 Operation

On transmit, data can be piped from one of three locations; host memory, shared memory, or transmit CPU memory into the network port, while the transmit CPU supervises transfers and compiles headers. No intermediate buffering of the application data takes place. We believe this is key to high speed operation.

On receive, parallel data from the network is pipelined to the host memory, local receive CPU memory, or shared memory. In normal operation, the receive CPU will DMA the header to local memory, perform initial processing to establish header integrity and payload destination, and then start a DMA process to transfer the data segment of the packet either to host memory or to the shared memory region. While storage of the payload is proceeding, the receive CPU completes its processing of the header information. Messages are passed between the transmit/receive CPUs and the host either by using the shared memory region or by using interrupt mechanisms which exist among all CPUs (host, transmit or receive).

The Protocol Accelerator enables a rapid data flow to and from the network by introducing a high degree of concurrency into the communication mechanism. Several activities execute simultaneously:

1) host processing (of higher layers and application),
2) transmit protocol processing,
3) receive protocol processing,
4) data transfer from host memory to the network adapter,
5) data transfer from the network adapter to host memory,
6) receive data checksumming,
7) transmit data checksumming, and
8) MAC frame processing.

2.4 Direct Memory Access

An important feature of the hardware architecture is the dual direct memory access controllers (DMACs). The DMACs are capable of moving 32 bit data words at rates of up to 33 Mbytes/sec over VME or 30 MBytes/sec over the VSB bus directly to and from the network ports. Scatter-gather type operations are fully supported both to and from the application memory. Data paths also exist for DMA of data between the network ports and any other RAM area on the PA (i.e. CPU RAM space or shared RAM space) so that intermediate data buffering is possible whenever necessary.

2.5 Checksumming

The on-board CPUs are capable of checksumming data at a rate in the region of 75 Mbits/sec[13] but operation at this rate would leave no time for protocol processing. To maximize our data throughput, it was decided to include hardware

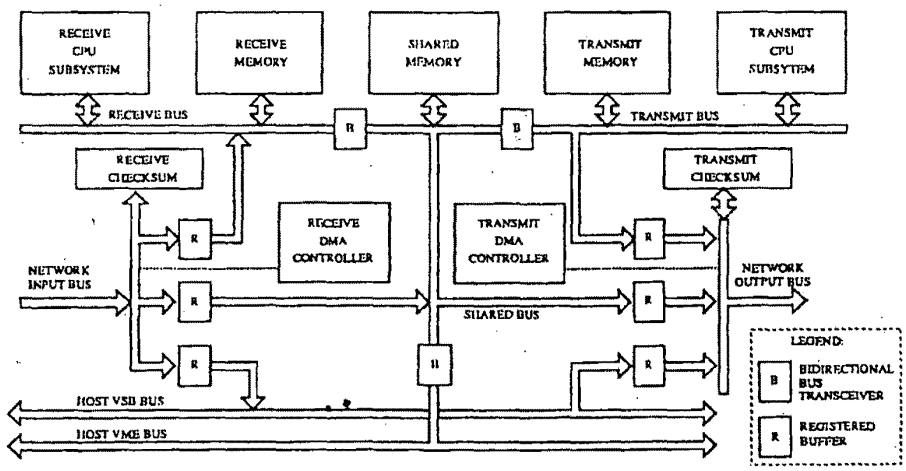

FIGURE 2. Protocol Accelerator Functional Block Diagram

checksumming in preference to using faster, more sophisticated processors for this function.

We use an 'on-the-fly' technique for the checksum and in order to take advantage of this hardware, the checksum field is required to trail the fields being checked. In the case of TCP this requires that the checksum field be moved from its usual place in the header. Although hardware schemes can be devised which leave the checksum field in place, these are somewhat more difficult to implement, and because the intent with this design was to produce a testbed suitable for many protocols, the checksumming circuitry was not designed to be protocol specific.

The circuits utilize 32 bit ones complement adders and operate in tandem with the DMA controllers; every word moved by the DMA controller is simultaneously applied to the checksum circuits. On the transmit processor, the circuit automatically inserts a 32 bit checksum at the end of both the header and the data segments. On the receiver these fields are checked, again automatically.

3. TCP IMPLEMENTATION

3.1 Introduction

The transport protocol used to demonstrate the capabilities of the Protocol Accelerator is the Transmission Control Protocol (TCP). It was chosen because it is currently one of the most wide-spread transport protocols in use on networks today. It is also the subject of a great deal of continuing research, and the expanding use of UNIX-based desktop workstations is increasing its penetration. Along with the increased use of TCP is the fear that as network rates rise, network throughput may not keep pace or may even decline as connection-oriented, window flow-controlled protocols such as TCP become a bottleneck. Some researchers do not believe that the protocol is to blame for the low throughput observed when current TCP implementations are run over experimental high speed networks [2]. Instead, they cite inefficient implementations of the protocol and interactions with the host operating system (UNIX) as the causes of the poor performance. The intensity of this debate will grow as more high performance machines running TCP observe less than ideal performance over high speed networks. Therefore, TCP was implemented to show that with an efficient implementation on the proper hardware platform, even a protocol designed for moderate-speed, error-prone networks can achieve high throughput.

3.2 Implementation Details

The high performance aspects of the Protocol Accelerator such as the dual processors, DMA, and on-the-fly checksum require the design of the transport protocol implementation to be specific to the PA. Therefore, a custom implementation of TCP was developed. The modules were written in C for the main protocol processing functions with embedded 68020 Assembly language code to perform many of the hardware specific tasks such as setting up the DMA controller, controlling timers, and polling status flags. No operating system is used on the PA. Many implementation decisions were made to optimize speed and efficiency for the 'main path' of protocol processing at the expense of processing for infrequent error conditions. These decisions are justified given the low error rates characteristic of high speed fiber networks.

The dual processor hardware architecture of the Protocol Accelerator leads naturally to the software architecture. The TCP implementation consists of separate transmit and receive processes running their respective tasks on separate microprocessors. The transmitter and receiver do most of their processing on data stored in private memory, but they do communicate through the shared memory. The bulk of this communication occurs through the Transmission Control Block (TCB), the main TCP state information structure which resides in shared memory. At appropriate times, state changes in either the transmitter or receiver are updated in the TCB which may then be read by the other processor in the course of its work.

As noted in the hardware description, the generic nature of the PA requires that checksums be placed after the header and after the data. Other than this difference, the implementation provides all of the TCP functions required to transmit and receive data in the TCP (call) ESTABLISHED state. Among others, these functions include maintaining the retransmission queue, providing resequencing for out of order packets, supporting retransmission timers, and packetizing host data into TCP segments, or TPDUs. It must be noted that to minimize data movement, host data is moved directly between host memory and the network interface. This precludes further segmentation/reassembly of the data at what would be the Internet Protocol (IP) layer. Therefore, although certain functions of the IP layer are rolled into the TCP header (IP address, length, protocol), IP functionality is not claimed.

Another objective of this work is to quantify the effects of the end host system on outboard protocol processing. To this end a UNIX device driver, applications programming interface (API), and application were developed for the host UNIX system. The relationships among the components are shown in Figure 3.

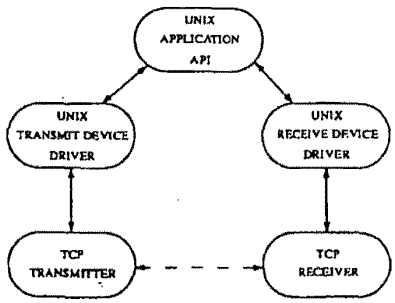

FIGURE 3. System Software Architecture

Because many variations are possible with software in the UNIX environment, attempts were made to keep the device driver, API, and application code as simple as possible while maintaining functional similarity to current methodologies for protocol/system interfaces. The results may then be used to extrapolate meaningful performance expectations of other systems with different basic parameters such as processor capability, network packet size, or bus speeds.

4. PERFORMANCE MEASUREMENTS / ANALYSIS

4.1 Test Configuration

For these initial results, the circuit was operated in a hardware loopback configuration (see Figure 1) with the network output port connected to the network input port via FIFO buffers. The loopback causes no loss, errors, or reordering of packets and is thus a best case. For the host we used a VME based single board computer based on the Motorola 68030 processor operating at 25 MHz. This processor was equipped with an area of shared dynamic RAM (4 MBytes) accessible from both the host processor subsystem and the VME bus.

The communications model used to test the capabilities of the Protocol Accelerator is that of a file server/client which can either provide or receive large messages at the throughput rates of the PA. Performance measurements were taken based on the UNIX host transmitting bulk data over the network which, in this system, is a loopback to the receiver side of the PA. In the initial measurements we found the transmitter to be the rate-determining element in the end-to-end process, being roughly fifty percent slower than the receiver. We therefore focus our discussion on the transmitter. In order to fully test the capabilities of the transmitter, the received data is not run back to the UNIX host because that would cause excessive contention for the VME bus. For these measurements only one host bus (VME) was used.

4.2 Protocol Accelerator Performance

Figure 4 shows the timing diagram for the flow of TPDUs through the PA transmitter. The header generation time is the time taken for the processor to access the current state information and populate the header fields. Once the payload DMA transfer is underway, further transport protocol processing proceeds in parallel and does not impede data transfer unless the payload transfer time is shorter than the overall transport processing time.

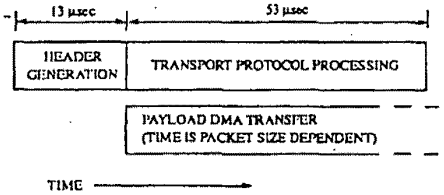

FIGURE 4. Transmitter Per Packet Processing Time

The payload transfer time is simply the product of the payload size (in 32 bit words) and the data transfer cycle time. The data transfer cycle time for our system can be calculated by adding the minimum DMA cycle time of 80 nsec (asynchronous) to the source (or destination) memory access time. The shared PA static RAM has an access time of 40 nsec, leading to a cycle time of 120 nsec per 32 bit word (i.e. peak data transfer rate in excess of 250 Mbits/sec). The VME memory on the available host computer, on the other hand, has a measured response time averaging 580 nsec, which reduces the peak throughput to below 50 Mbits/sec when this memory is used. These figures assume no other activity on the system bus. For these preliminary findings we use a constant packet size of 2016 bits of which 1728 bits was payload. The total transport processing time of our transmitter TCP implementation is 66 µsec and this becomes the limiting throughput factor for small payload packets. The header generation part of this time is 13 µsec.

Using these results, it is straightforward to extrapolate the performance of the PA transmitter for any memory access time and packet size. An example is shown in Figure 5. Here, the payload throughput and the packet throughput are plotted against packet size for different memory access times. The flat region of the packet throughput curves are in the region where the protocol processor limits throughput. To the right of this region, throughput is limited by memory bandwidth. Two cases are shown, 120 nsec and 660 nsec memory cycle times; these represent the memory cycle times for the two cases described previously, PA shared SRAM and host VME DRAM.

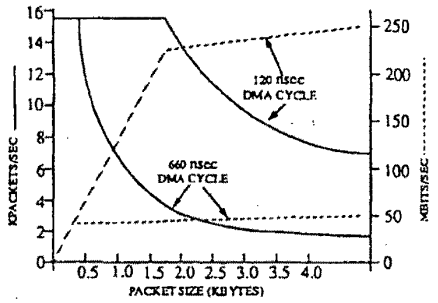

FIGURE 5. PA Throughput Based on Memory Access Time and Packet Size

4.3 PA Performance in UNIX Environment

We now examine the effect of host system software on the observed throughput to the application. Host system interactions with outboard processors are critical because regardless of how fast the data may depart from or arrive to the outboard processor, it is not until the data is actually in user space on the host system that the communication is complete.

The UNIX overhead time was measured for a number of host message sizes transmitted from the PA. The UNIX overhead consists of the time between the user call to the API's BSD-like write socket call and the receipt of the command by the PA on the inbound side plus the time between the generation of the UNIX interrupt (signaling the end of PA processing of the host message) and return of control to the user at the end of message transmission. The connection establishment functions of socket and connect are not included in these measurements because they are not directly associated with the per message data transfer. The system was kept unloaded except for standard system daemons in order to minimize contention for processor and bus resources. The results of the UNIX overhead measurements for message transmission are shown in Figure 6.

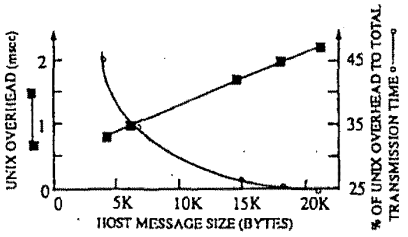

FIGURE 6. Absolute and Relative UNIX Overhead vs. Host Message Size

As the message size increases, the absolute UNIX overhead increases while its percentage of the total operation time falls. If the TCP interface process had been the only process running on the host system, the absolute overhead would remain constant as host message size increases. This is because the only action required of the host is to change the command message length delivered to the transmitter. The observed results are due to the accumulation of higher priority host processes while the PA DMA is in control of the UNIX memory. As the message size increases, the DMA controls the host bus for more time per message. The linear increase in absolute overhead also indicates that the higher priority functions are generally periodic. It should be noted that even as the absolute overhead increases with message size, the on-board TCP processing time remains constant on a per segment basis. This shows that the increase in absolute overhead is not due to increases in time waiting for the host bus or other per segment processing; it occurs at the end of the packet processing before control is returned to the user process.

The declining percentage of overhead as host message size is increased confirms our expectations. As the host message size increases, the amount of time spent on UNIX overhead processing relative to the overall transmission time decreases. This again is due to the constant amount of code which must be executed. The observed asymptotic approach to 25% represents a balance between the decreasing percentage of host processing time per message and the increased amount of system backlog which is serviced before the user process regains control. Figure 7 shows the overall performance of the PA in and out of the UNIX environment.

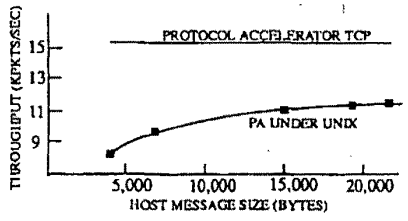

FIGURE 7. Throughput vs. Host Message Size

5. CONCLUSIONS

An outboard protocol accelerator optimized for transport layer processing has been implemented and tested. The design consists of separate transmit and receive units, each with its own CPU, checksum, and DMA hardware. Using an implementation of TCP, we demonstrated that this approach can support end to end throughput rates in excess of 15,000 packets/sec between network and application. When integrated into the UNIX environment, the protocol accelerator freed the host of the traditional protocol processing tasks but led a 25% decrease in average throughput due to the UNIX overhead.

ACKNOWLEDGEMENTS

We would like to thank Mike Koblentz for his major contributions and Michael Stanton for his part in the initial design of the hardware.

REFERENCES

[1] L. Svobodova, "Measured Performance of Transport Service in LANs", *Computer Networks and ISDN Systems*, vol 18, pp. 31-45, 1989/90.

[2] D. D. Clark, V. Jacobsen, J. Romkey and H. Salwen, "An Analysis of TCP Processing Overhead", *IEEE Comm. Mag.*, pp 23-29, June 1989.

[3] D. D.Clark, M. L. Lambert and L. Zhang, "NETBLT; Bulk Data Transfer Protocol", Network Information Center RFC-998, SRI International, Menlo Park, CA, 1987.

[4] G. Chesson, E. Brendan, V. Schryver, A. Cherenson and A. Whaley, "XTP Protocol Definition." Protocol Engines Inc., 1900 State St., Santa Barbera CA. 93101.

[5] R. P. Singh and A. Erramilli, "Protocol Design and Modeling Issues in Broadband Networks", *Proc. ICC '90*, Atlanta, GA, pp 1458, 1990.

[6] D. R. Cheritan, "VMTP; Versatile Message Transaction Protocol, Protocol Specification", Network Information Centre RFC-1045, SRI International, Menlo Park, CA, 1987.

[7] A. N. Netravali, W. D. Roome, K. Sabnani, "Design and Implementation of a High-Speed Transport Protocol", *IEEE Trans. on Communications*, vol 38, no. 11, pp.2010, Nov. 1990.

[8] D. Giarrizzo, M. Kaiserswerth, T. Wicki, R. C. Williamson, "High Speed Parallel Protocol Implementation", H. Rudin and R. C. Williamson, *Protocols for High Speed Networks*, North-Holland, 1989.

[9] M. Zitterbart, "High Speed Transport Components", *IEEE Network Magazine*, January, 1991, pp. 54-63.

[10] H. Kanakia and D. R. Cheritan, "The VMP Network Adapter Board (NAB); High performance Network Communication for Multiprocessors," *Proc. ACM SIGCOM '88 Symposium on Communications Architectures and Protocols*, Stanford University, CA, 1988, pp. 175-187.

[11] E. C. Cooper, P. A. Steenkiste, R. D. Sansom, B. D. Zill, "Protocol Implementation on the Nectar Communication Processor", *Proc. SIGCOM '90*, Philadelphia, PA, 1990 pp. 135.

[12] N. Jain, M. Schwartz, T. R. Bashkow, "Transport Protocol Processing at GBPS Rates", *Proc. SIGCOM '90*, Philadelphia, PA, 1990, pp. 188.

[13] R.Braden, D. Borman, C. Partridge; "Computing the Internet Checksum", Network Information Center RFC-1071, SRI International, Menlo Park, CA, 1988.