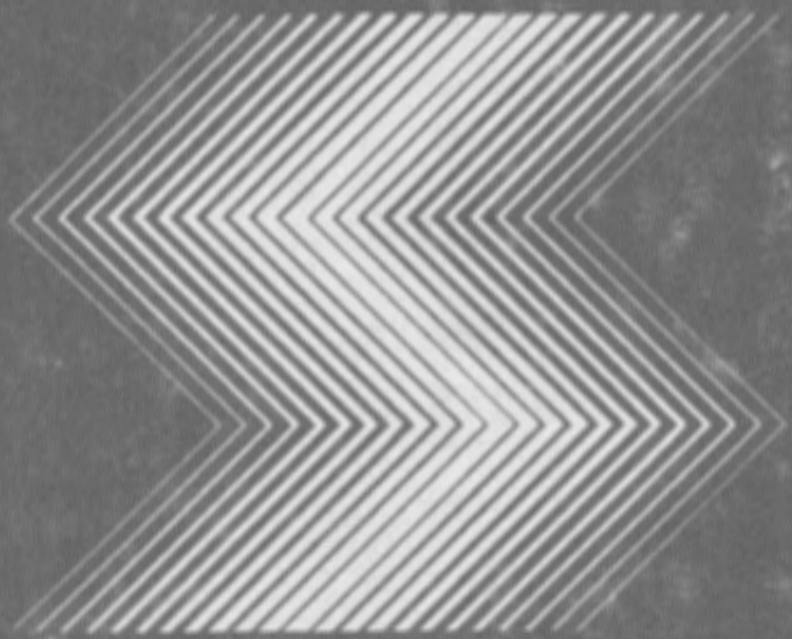
Computer Communication Review
Volume 18, Number 4
August 1988

acm PRESS

SIGCOMM '88 SYMPOSIUM

Communications Architectures & Protocols

Stanford, California
August 16–19, 1988



SPONSORED BY ACM SIGCOMM WITH SUPPORT GIVEN BY THE INFORMATION SCIENCES AND TECHNOLOGY CENTER OR SRI INTERNATIONAL.

The Association for Computing Machinery
11 West 42nd Street
New York, New York 10036

ISBN O-89791-279-9

Additional copies may be ordered prepaid from:

ACM Order Department
P.O. Box 64145
Baltimore, MD 21264

Price:
Members $20.00
All others $26.00

ACM Order Number: 533880

The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors

Hemant Kanakia
Computer Systems Laboratory
Stanford University

David R. Cheriton
Computer Science Department
Stanford University

Abstract

High performance computer communication between multiprocessor nodes requires significant improvements over conventional host-to-network adapters. Current host-to-network adapter interfaces impose excessive processing, system bus and interrupt overhead on a multiprocessor host. Current network adapters are either limited in function, wasting key host resources such as the system bus and the processors, or else intelligent but too slow, because of complex transport protocols and because of an inadequate internal memory architecture. Conventional transport protocols are too complex for hardware implementation and too slow without it.

In this paper, we describe the design of a network adapter board for the VMP multiprocessor machine that addresses these issues. The adapter uses a host interface that is designed for minimal latency, minimal interrupt processing overhead and minimal system bus and memory access overhead. The network adapter itself has a novel internal memory and processing architecture that implements some of the key performance-critical transport layer functions in hardware. This design is integrated with VMTP, a new transport protocol specifically designed for efficient implementation on an intelligent high-performance network adapter. Although targeted for the VMP system, the design is applicable to other multiprocessors as well as uni-processors.

1 Introduction

Performance of transport protocols on multi-megabit communication networks tends to be limited by overhead at both the transmitter and receiver. For example, measurements of the V kernel [5] indicate that network transmission time on Ethernet accounts for only about 20 percent of the elapsed time for transport-level communication operations, even with its highly optimized protocol. Similar performance figures have been reported in [15, 17]. Although processor and memory cycle times keep improving, with communication networks moving to gigabit range, we expect the processing to persist as a bottleneck - unless significant improvements in network adapter and transport protocol designs are achieved. We identify three major problems with current designs.

First, the host-to-network adapter interface imposes excessive overhead, particularly on a multiprocessor host, in the form of processor cycles, system bus capacity and host interrupts. The processing overhead arises from calculating end-to-end checksums, packetizing and depacketizing as well as encryption, in the case of secure communication. The memory-intensive processing required by these functions reduces the average instruction execution rate especially for a high-performance processor such as MIPS [14] in which memory reference operations are proportionally much slower than register-only operations. This processing causes the data to move at least twice over the system bus; once from global memory to the processor (or its cache), and once when the packet is copied to a network adapter. The increased traffic wastes system bus bandwidth, a critical resource in multiprocessor machines. In current host-to-network adapter interfaces, a host is interrupted for each packet received or transmitted. These per-packet interrupts force frequent context switching with the attendant overheads, with a high penalty in the multiprocessors system with processor caches, where it may be necessary to fault code and data into the cache before responding to the interrupt. In addition, the context switch may also incur contention overhead when data associated with the network module is resident in another cache. The problem is further aggravated by the prospect of networks moving to the 100 megabit up to the gigabit range using fiber optics [11, 13, 16]. For instance, in a file server attached to a 100 megabit network with the interface interrupting on every 2 kilobyte packet, the network interrupts every 200 microseconds under load, hardly sufficient to do even minimal packet processing.

Second, the so-called "intelligent" network adapters that implement transport-level functions have lower performance at the transport-level as compared to the alternative system where a network adapter does programmed I/O transfers and a host performs transport protocol functions. The primary reason is an inadequate internal memory architecture. Currently, the data transfers into and out of the buffer memory reduces the number of memory cycles available for packet processing. The future system bus technology, with a high transfer rate and the burst-mode transfer, and the future networks, with a high data rate, will make this problem even more acute.

Finally, conventional transport protocols are too complex or awkward for hardware implementation and too slow without it. For a large packet, the processing cost incurred in checksumming and encryption dominates the packet processing, since the cost increases in proportion to the size of a packet. Hardware implementation for such key performance-critical functions would substantially increase performance, but the packet formats of

conventional transport protocols are does not facilitate hardware support or implementations.

An additional factor that motivates rethinking network adapter architecture is the problem of a host being bombarded by packets from one or more other hosts. The packet arrival rate, especially in a high-speed network, can exceed the rate at which a host can process and discard these packets, effectively incapacitating the host for useful computation. Excessive packet traffic can arise either from failures or malicious behavior of a remote host. A well-designed network adapter acts as a "firewall" between the network and the host.

In this paper, we describe the Network Adapter Board (NAB) for the VMP multiprocessor [3], focusing on the architectural issues in the host interface, the adapter board, and the transport protocol. The adapter host interface architecture is designed for minimal latency, minimal interrupt processing overhead and minimal data transfer on the system bus. The NAB uses a novel internal memory and pipelined processing architecture that implements some of the key performance-critical transport layer functions in hardware. The design is coupled to a new transport protocol called *Versatile Message Transaction Protocol (VMTP)* [1]. VMTP is a request-response transport protocol specifically designed to facilitate implementation by a high-performance network adapter. VMTP assumes an underlying network (or internetwork) service providing a datagram packet service, and includes support for multicast, real-time, security and streaming. The interested reader is referred to an Internet RFC [1] for further details. For brevity in this report, we focus on aspects of the protocol relevant to the cost of communication and the design of the VMP network adapter. We describe the expected performance of the NAB prototype being built. Although targeted for the VMP system, the design is applicable to other multiprocessors as well as uni-processors.

The next section describes the host-to-network adapter interface architecture. Section 3 describes the internal architecture of the network adapter board. Section 4 describes details of a prototype of the network adapter. Section 5 indicates the (expected) performance for this adapter. Section 6 compares our design to related work on this problem. We conclude with a summary of the current status, our plans to further evaluate this design and the current open problems in this area.

2 Host-Network Adapter Interface

A request/response model of communications is used for information exchange between a network device and a host. A host requests network services with a control block sent to device, and the device returns a response after the service is provided. These messages are transferred across via an interface that appears to the host software as a 1024-byte control register.

To transmit data, the host software writes a control block, the *Transmit Authorization Record (TAR)*, to this control register. The TAR contains control information describing data to be sent including the pointer to data in physical memory. If the data fits entirely within the control register, the data segment description is omitted from TAR. In both cases, the network adapter transmits the data, checksumming and encrypting (if required). Interface interrupts host to inform that a TAR has been used to successfully transmit the data.

To receive data, the host software writes a control block, a *Receive Authorization Record (RAR)*, that "arms" the network interface to receive packets, specifying the maximum size to receive and providing a list of pointers to memory pages in host memory into which to deliver the data. The RAR can specify any one of: (1) a specific source, (2) a class of allowable sources or, (3) any source for the received data. where source is either a transport-level or network-level address.

The interface interrupts the host when the received packet(s) satisfies one of these RARs, returning the RAR, along with the packet header, to the appropriate host via the control register. The returned RAR itself may contain small amounts of data in addition to the corresponding packet header. When the RAR is returned, the data has been already stored in the host memory at the location pointer(s) contained in it, unless the data is contained in the RAR. Incoming packets are discarded if they cannot be matched to an outstanding RAR.

Type, a subfield in the control block, distinguishes the records passed via the control register. Four major types of records, used in transmission and reception of data, are an RAR with small amounts of data, an RAR with data descriptors, a TAR with small amounts of data and a TAR with data descriptors. To optimize host-network adapter interactions, host is allowed to combine issuing of a TAR and RAR, using the same buffers to send and to receive data. Additional types of records are used by a host for various purposes such as to add or delete acceptable destinations, to restrict traffic from a source, to provide decryption/encryption keys to the NAB, to get status information, and to reset the NAB.

The RARs and TARs include a packet header including a network-level header, either small amounts of data or a list of data descriptors, and various control information. The buffer descriptors in a TAR point to locations in the physical memory space where data to transmit is available. The buffer descriptors in an RAR point to locations in the physical memory space where the data is to be received. The returned RAR or TAR contain in addition to the buffer descriptors the number of data words actually received or transmitted. The control information, used by NAB, includes a link field, type of RAR matching to be used, transport-level source and destination addresses, interrupt control, timeout control, a local host number, and a local process identifier. The buffer descriptors are omitted for TARs and RARs containing small amounts of data. A link field, used by the adaptor, allows chaining of these records as necessary. The type of RAR matching to be done indicates if the RAR can be used for receiving traffic only from the specified source or any source.

Interrupt control is used to determine when to interrupt the host identified in the record. On reception, the host, indicated by the host number, is interrupted either when the data of the first packet is stored in the system memory or when the data is completely received or both. The completed RAR is returned via the control register with the length of data received before the host indicated in RAR is interrupted. On transmission, one may have the host interrupted either when the NAB begins processing a TAR, or when the last of the data segment is transmitted. The TAR is written into the control register before a host is interrupted.

In the following, we discuss how this interface efficiently handles small amounts of data, large amounts of data, and also allows the interface to act as a firewall, protecting the host from the network.

2.1 Short Message Handling

The latency with short messages is minimized because the short message is written to the interface as part of the TAR and read as part of the returned RAR on reception. The operating system interrupt handlers for the network adapter can directly copy the message data between the control register and the operating system data structures, moving the higher-layer data to its intended destination with minimal cost. For multiprocessor hosts with per processor cache this procedure also avoids additional cache and bus activity, as we expect the small amount of data to be already available in cache before being sent with TAR or used immediately after having received in an RAR. Thus, the delay introduced in transmitting and receiving a packet with a small amount of data is no more than that incurred with a host directly handling the packet and using the interface as a staging area to send and receive packets.

Note that including the packet header in the TAR means that the processor writes a small amount of data to the interface for transmission yet the network adapter has minimal processing on the data to prepare packets for transmission. In particular, for small data appended to header, the network adapter need not do anything before starting network transmission, given that checksumming and encryption occur as part of transmission.

2.2 Long Message Handling

Host overhead is minimized for the transmission and reception of large amounts of data, typically in the range of 4-16 kilobytes, by passing descriptors rather than actual data. On transmission, the host writes one TAR and receives one completion interrupt, with the network adapter transferring the data from host memory with minimal bus overhead.[1] The network adapter handles the per-packet overhead of packetizing, checksumming, encryption and per-packet coordination. On reception, the host receives a single interrupt for each RAR returned after the data has been transferred into global memory. Again, the per-packet interrupt overhead is handled by the network adapter.

Latency in transfer of large amounts of data is reduced by ensuring minimal bus and memory references. Moving byte-based processing functions such as checksumming and encryption to network adapter reduces memory references to one per data word. Passing buffer descriptors to network adapter which retrieves data from host memory ensures that only one bus transfer per word transferred is required. For multiprocessors with per-processor cache, the buffer-passing model helps reduce number of cache misses one would otherwise incur, as a cache is neither likely to have most of the data transferred nor going to use it in the near future. The cache pollution, resulting from using cache for network data transfers, would also increase cache miss ratio for other applications. An additional factor reducing latency is the use of burst-mode transfer on host bus, which decreases transfer time of data on the bus by a factor of 4.

In sending and receiving groups of packets, the interface can afford to introduce some latency for the first packet of the group as long as the whole transmission and reception has less delay as compared to a host processor handling per packet processing. That is, for a small number of packets K, it should be the case that

$$K * P_{host} > D + K * P_{interface}$$

where writing the control record to the interface introduces a delay D in transmission over the host processor writing the data directly to the network, K is the number of packets to be transmitted, P_{host} is the time for the host to packetize and send one full-sized packet and $P_{interface}$ is the time for the interface to transmit one full-sized packet. The value of K for which this is true should be as small as possible, ideally 1 but certainly less than the common size of a multi-packet packet group. In this interface, the value is close to 1 primarily because $P_{interface}$ is much smaller than P_{host}.

[1] The VMP memory supports block transfer using the VME serial bus protocol, thereby minimizing bus occupancy and arbitration overhead.

2.3 Network Firewall

The interface architecture is designed to allow the network adapter to function as a *firewall*, protecting the host from network packet pollution, both accidental and malicious. In essence, a host incurs no overhead for network packets whose reception it has not authorized; the interface discards all packets that are not compatible with an RAR provided by the host and are not directed to an end-point registered with the adaptor. Some examples of its use follows. If the host does not provide an RAR for broadcast packets, then garbage broadcast packets incur zero overhead on the host processor(s). Multiple responses generated by a multicast request can be limited to only those that fit into the buffer area provided; the rest are discarded without incurring any host overhead. By providing RARs for only those sources it wants to listen to, a process could avoid host overhead required otherwise for pruning out the traffic from unauthorized sources. In general, the authorization model of packet reception plus the speed of the network adapter insulates the host from packet pollution on the network.

3 Network Adapter Internal Architecture

The network adapter internal architecture is designed to provide maximal performance between the host interface and the network architecture. The internal architecture is structured as five major components, interconnected as shown in Figure 1. These components serve the following functions:

Network access controller (NAC) : implements the network access protocol and transfers data between the network and the packet pipeline.

Packet Pipeline : generates and checks transport-level checksums and performs encryption and decryption of data in secure communication.

Buffer memory : a staging and speed-matching area for data in transit between the packet pipeline and the host memory. Its specialized buffer memory permits fast block data transfers between the network and host and provides the on-board processor with contention-free memory access to the packet data.

Host block copier : moves data between the buffer memory and the host memory using a burst-transfer bus protocol, minimizing the latency as well as the bus and memory overhead for transfers.

On-board processor : a general-purpose processor that manages the packet processing pipeline and various bookkeeping functions associated with the protocol.

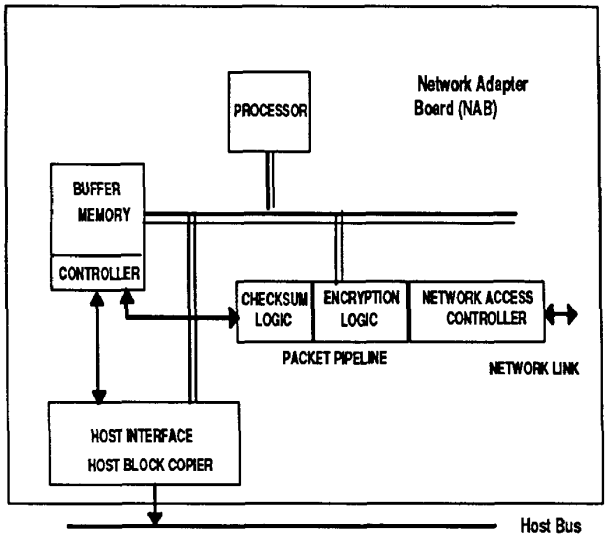


Figure 1: Network Adapter Internal Architecture

Transmission is handled in three steps. When a Transmission Authorization Record is written to the interface, it is moved into the interface from the host memory by the host block copier. The TAR provides a description of the segment of data in the host memory to transmit. Next, the on-board processor forms the first packet from the TAR and first data blocks of the message and queues the packet for the packet pipeline using the information provided by the TAR. Finally, the packet is processed and transmitted by the packet processing pipeline and NAC at the network data rate. The pipeline calculates a checksum and optionally encrypts the data as the packet is transmitted.

On reception, a packet is accepted by the NAC and passed through the packet pipeline which decrypts the data, verifies the checksum, and deposits the received packet into the buffer memory. If the received checksum is verified, the packet is matched to the appropriate RAR. For the first packet in a group of packets, this may involve locating and allocating a non-specific RAR, making it dedicated to receiving more packets from this source. The packet data is then delivered into the host memory associated with this RAR. The reception of a packet into the buffer memory proceeds concurrently with both the checksum verification and the transfer to the host of previous packets. On reception of the final packet or on timeout, the host is interrupted and informed of the receipt of this packet group by returning the RAR in the interface control register. Thus, the host is interrupted only once per RAR used by the NAB.

If there is no matching RAR for a packet, the adapter determines whether the destination address is locally acceptable. An address is *locally acceptable* if the address is in the local host's list of destination addresses, both individual and group addresses. The adapter then transmits a response to the local host indicating that the packet was discarded. When the destination address is valid but no RAR is found, the interface discards the segment data and returns an indication to the destination host via the control register. The interface does not time out a partially-filled RAR; the partially-filled RAR is returned to the host only on an explicit request from a host. The host handles sending out retransmission requests and various timeouts. The host also sends acknowledgments and handles retransmission requests by issuing a new TAR.

Three key aspects to the design are the buffer memory, the packet processing pipeline, and the use of the general-purpose processor, as discussed in the following sections.

3.1 The Buffer Memory

The network adapter requires buffer memory in order to speed-match between the host bus and the network as well as to provide a staging area for the transmission and reception of packets. Three issues arise with the buffer memory design. First, the buffer memory must provide sufficient contention-free memory access to support simultaneous uses by the on-board processor, NAC, and block copier, as occurs under load. The performance of many so-called "smart" network adapters suffer from this contention. Second, the buffer memory must minimize latency for packet transmission and reception over direct transmission between the host memory and the network. Finally, a provision is required to prevent overcommitting the buffer memory to either transmission or reception, which would interfere with the functioning of the adapter. Our approach to each of these issues is discussed below.

3.1.1 Buffer Memory Contention

To minimize contention, the buffer memory uses dual-port static column RAM components, also referred to as Video RAM ICs. The Video RAM-based buffer memory, shown in Figure 2, provides multiple buffers to hold and to process packets while a packet is being received or being transmitted. This IC provides

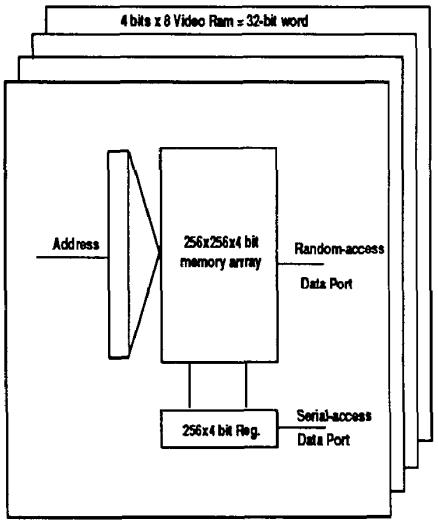


Figure 2: Video RAM-based Buffer Memory at Network Adapter Board (NAB)

two independently accessed ports: one providing high-speed burst-mode transfer, and the other providing random-access. The serial-access port is used to move a packet from the network into

the buffer memory or the data from host memory to the buffer memory. The random-access port provides memory access for on-board processing of packets by the adapter's general-purpose processor. The serial access does not need address set-up and decoding time so read-write times on this port are faster than read-write times for a RAM array. For instance, in our prototype, memory is 32 bits wide, the serial access time is 40 ns per word and the cycle time for random read/write access is 200 ns. This gives an effective transfer rate of 800 megabits/second over the serial port and 160 megabits/second over the random-access port. To provide the equivalent memory bandwidth on a single ported standard 32-bit wide memory would require a memory IC with read/write cycle time of 33 ns. Currently, such fast memories are available, but they cost more and have less memory density than video RAMs.

Data transfer operations in this memory proceed as follows. A packet (or data) is received from the network (or the host) via the serial port into the shift register contained in Video RAM ICs. The shift register acts as the temporary storage. When the block is completely received, it is transferred from the shift register, in a single memory cycle, to a row of the memory cell array constituting the buffer memory. The processor manipulates header fields of packets stored in the array via the random-access port. The processing of a packet continues without interference while the next packet is being received, except for one memory cycle stolen for each received packet transferred to the memory cell array from the shift register.

Video RAM ICs provide performance that closely approximates the performance of true multiport memories, but at a fraction of the cost. A triple-port memory cell would triple the area of the memory cell, reducing memory density and increasing its access time. Video RAMs provide full memory bandwidth to the processor at low cost. They also allow high-speed block data transfer between the buffer memory and the host and the network. The separate serial port avoids the processor losing memory bandwidth to arbitration overhead on the random access port. The serial transfer ports, accessed in parallel across a bank of video RAM ICs, maximizes the data unit to be transferred per arbitration between the host block copier and the NAC and minimizes the transfer time, thereby minimizing the arbitration penalty.

High speed FIFOs could be used as an alternative to the buffer memory described above to speed-match between the host and the network. Intuitively, a FIFO-based design should have minimal delay for two reasons. With short FIFOs, one begins to process and transmit the packet before the entire packet has been copied. With FIFOs, one avoids the software overhead of managing input and output packet queues. Nevertheless, our buffer memory design provides better performance for reasons outlined below.

When the system bus is lightly loaded, our design compares favorably with the FIFO approach for large and small amounts of data transfers. For large data transfers, our design amortizes the buffering latency of the first packet over multiple packets. For a short single packet transfer, the difference in delay is small because the main source of delay in this case is the processing done at a host and the adapter, not the time of copying data to the interface.

However, at even moderate levels of bus traffic, our design outperforms the FIFO-based approach. When the bus is congested, the demand for bus access may not be satisfied in time to avoid underrunning or overrunning a FIFO, resulting in packet loss at a sender and at a receiver. The time thus lost in transmitting aborted packets as well as the retransmission delay increases the total time needed to transfer a data segment.

Mismatch between transmission data rates on the host bus and the network channel also supports using the buffer memory. When host bus speed is much higher than network channel speed, the additional delay for bringing the first packet in full before beginning transmission is negligible compared to the total transmission time for a large data segment. When host bus speed is comparable to (or lower than) the network channel speed, bringing in a full packet before starting transmission is necessary to avoid (frequent) loss of packets by contention on the bus.

3.1.2 Buffer Latency

Buffer latency refers here to the additional delay caused by the on-board buffering of a packet over the direct transmission by host to network. We minimize it by using a hardware block copier and by using contention-less memory accessing. A block copier transfers data between host memory and NAB using a serial memory transfer protocol, which minimizes transfer time and bus occupancy. Moreover, the NAB memory design described above facilitates high speed block data transfers [2] and provides contention-less memory accessing, both minimizing the buffering latency.

In this design, the cost of the buffering latency of the first packet is amortized over the subsequent packets sent in the packet group. The subsequent packets are copied from the host memory or the network link in parallel with the processing of the previous packet transferred, hiding the cost of their buffering latency. For large amounts of data, the buffering latency of the first packet forms only a small fraction of the total delay. For instance, the minimum transmission time for 16 Kbytes of data over a 100 megabits/sec network is 1.31 ms; whereas, the buffering latency for the first packet in this packet group is approximately 25 microseconds, [3] which is less than 2 percent of the total transmission time. The latency is even less significant compared to request-response delay for large data transfers, as this delay includes processing times at the host processors.

We note that the data transfer between host and interface buffer memory does not constitute an extra copy because the NAB performs all the functions that the host processor would have otherwise performed if it was performing the transfer. That is, the interface memory is required as a staging area for incoming and outgoing network traffic in any case.

3.1.3 Reception of Garbage Packets

Ideally, the mechanism for matching a packet to an RAR is fast enough so that it cannot be overrun by receiving garbage packets from the network. The packets with correct node address or multicast packets may still be undesirable, if the host is being bombarded with them by a network malfunction. In the interface, the on-board processor is not involved in transferring packets into the interface memory, and can receive a number of packets without being interrupted by the NAC. In the prototype, the RAR matching algorithm run at the NAB is fast enough to keep up

[2] The maximum transfer rate, using currently available Video RAMs, may be as high as 800 megabits/sec, assuming word width of 32 bits and 40 ns cycle time on the serial port.

[3] Calculated by assuming packet size of 1024 bytes and block transfer rate of 320 megabits, available with the burst transfer on the VME bus.

with a packet rate in excess of 16,000 packets/sec. As long as the network throughput does not exceed this limit, the buffer memory cannot become full of packets that do not match an RAR, and the availability of buffer memory to receive authorized packets remains independent of the network pollution that may be taking place.

3.2 The Packet Pipeline

The network adapter incorporates several hardware modules that constitute an interlocked *packet pipeline*, performing checksumming, encryption and network access on transmission, and the reverse on reception, all operating at the network data rate. Each stage in this pipeline is a simple operation, such as memory read/write, compare, and one's complement, as needed for these algorithms. The data transfer unit between stages is 32 bits. A word is prefetched at each stage and stored until the function is completed. A short FIFO is included in the NAC to allow for setup delays for decryption and node address recognition on packet reception. The on-board processor updates packet headers and queues the packets for transmission as well as processes packets on reception.

Use of the packet pipeline, mandates the common format used for all VMTP packet types, shown in Figure 3, which is designed so as to minimize processing delay, overhead and complexity. The first field specifies the client entity associated with the message transaction, the indicator for the decryption key to be used with the packet. The second field indicates the length of the packet and various address qualifiers. It is sent in the clear, providing the network adapter with some time to access the decryption key before it encounters the encrypted portion of the packet. Various other fields follow that indicate the VMTP type of packet, its packet group and its position within the packet group. The order of these fields also corresponds to the order in which the reception logic needs to map the packet to a particular client, packet group and data within the packet group. The checksum field is located at the end of the packet, (rather than in the header) so that it can be calculated "on the fly" on both transmission and reception. The length field specifies the offset of the checksum field from the beginning of the packet.

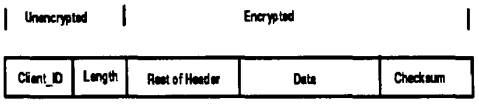


Figure 3: Common format for all VMTP packets

Reception processing proceeds as follows. When a packet starts to arrive at the network access controller, it checks the destination address and then forwards the first words of the packet to the first stage of the decryption hardware. At this stage, the first data word received is delayed until a decryption key is located in the decryption key cache (indexed using a hardware hashing function on the destination address). If the key is located, the data proceeds through the decryption stages, each one of them performing a function in the decryption algorithm.[4] If the key is not located or decryption is not needed, the data stream proceeds to the checksumming stage with the decryption stages disabled. The start and end points for checksumming of data bytes is specified by the offset values initialized by the user. Each data word is passed through a 1's complement sum to form two 16-bit partial checksums.[5] At the end of receiving a packet, a 32-bit checksum is calculated from the partial checksum and added to the buffer. The final step is to write the data word in the serial staging register. When the end of the packet is received, the data is moved to a receive buffer address provided by the serial-port controller. The empty buffer addresses are provided to the controller by the on-board processor. The received buffer is queued for header processing by the on-board general-purpose processor.

The received packet is lost if the buffer memory is full. A packet is also lost whenever the transmission is interrupted due to the lack of data in any stage of the pipeline. Both these unusual conditions require flushing and restarting of the pipeline, a task that is handled by the processor.

A packet pipeline as described increases the performance due to the following reasons. The processing overhead of performing functions such as checksumming and encryption is hidden in the copying operation, i.e., the copying of packets from the network to the buffer memory. Otherwise, additional memory references would be required to perform checksumming or encryption, wasting bus capacity as well as memory bandwidth, which are critical resources in a multiprocessor system (and in a network adapter). The latency due to the packet pipeline is kept low by using very few stages, and by restricting the width of the pipe. In the prototype, the pipeline uses fewer than 5 stages and a 32-bit wide data path. In contrast, if a larger unit of inter-stage data transfer, such as a full packet, were used, the time to fill up the pipeline would be larger, thus increasing the latency to transmit a packet. Finally, many simple stages, implemented in hardware, and running in parallel provide better performance than a single fast processor performing the same functions.

The fine-grained, tightly synchronized processing achieved by the packet processing pipeline suggests that comparable performance would be difficult, if not impossible, to achieve with multiple general-purpose processors working in parallel. In particular, with a single packet transmission or reception, the multiprocessor solution would perform at the level of a single processor, which is far below the performance of the NAB pipeline. With multi-packet transmissions and receptions, each processor has to wait for the reception of "its" packet before it can start processing the packet. The combination of waiting for these large units of data, the processing overhead per packet and the limited number of packets in a "blast" makes this approach unattractive.

A general-purpose multiprocessor solution is further hampered by the degree of correlation observed between successive packets received from the network. For example, the measurements of the VMTP traffic on a 10-MB Ethernet [4] show that statistically the packet currently being received belongs to the same message transaction to which the previous packet belongs. Because packets in the same message transaction share control block information at sender and receiver, this correlation results in interference among multiple processors for both memory ac-

[4] For the prototype, we have used a single cycle of product cipher algorithm that is similar to the DES standard.

[5] The complete checksum algorithm used is described in the specifications for VMTP [1].

cess and synchronization. Thus, a general multiprocessor architecture in which each processor works on a different packet appears significantly inferior to the pipelined architecture of the NAB.

We have placed the packet pipeline between the buffer memory and the network, forcing it to run at the network data rate. The alternative would be to place it between the host bus and the buffer memory. The alternative placement is undesirable on many counts. Burst transfer rates on host buses are much higher than network rates, forcing one to operate the pipeline at higher data rates. The synchronization of the pipeline operation with the bus is also more difficult because the interface typically involves transfer of data from memory as well as control information from one or more host processors. Moreover, on reception we do not know where to put data until the received packet is decrypted, further complicating the pipeline design. Last but not least, placing the pipeline at the host-end precludes strict implementation of a network firewall; data is transferred across the system bus before detecting checksum errors or that the packet is not wanted.

Most transport-level protocols preclude strict pipeline processing as described here because the checksum field is included in the header. As a consequence, the checksum has to be calculated before the packet header can be transmitted, preventing significant pipelining. VMTP appends the checksum to the end of the packet to allow pipelining. It also uses a 32-bit checksum for compatibility with the pipeline width, as compared to the conventional 16-bit checksums used in TCP.

It is the design of packet pipeline which causes the network adapter to be protocol-specific. It implements a specific checksum and encryption algorithm. Moreover, it uses the layout of VMTP packet to calculate various offsets to begin and end checksum operations. Adopting a common packet format shown in Figure 3, and using programmable control, the pipeline could be used for other transport protocols. The problems posed by variable network-level header, often found in Internet environment, can be dealt with by including a length of header field within this header.

Each stage of the pipeline logic is simple and needs to use very low-level of integrations. This has the advantage that as the networks move to a gigabit range, the pipeline's rate of execution can be speeded up by implementing it in Gallium Arsenide based logic circuits, which currently provides above 1000 gates in a package and operates in gigabit range.

3.3 The On-board Processor

The network adapter includes a general-purpose processor that manages the buffer memory, the host block copier and the packet processing pipeline. A general-purpose processor was chosen for these functions because they are significantly more complex than other functions, offered less performance benefit in a specialized hardware realization, and were less understood than the packet pipeline and block copier functions. The cost of processing performed by the NAB processor seems to be equally shared among various functions rather than being dominated by a few performance critical functions, thus reducing any potential benefit of the hardware realization. Moreover, using a general-purpose processor instead of a special-purpose protocol processor allows one to benefit from the continual advances in microprocessor technology.

With the general-purpose processor, we can program the basic queueing and dequeuing required to manage the buffer memory as well as any functions required by the protocol. However, we have taken care to partition the processing between this on-board processor and the host processors as follows. The on-board processor executes the "common case" packet processing, namely, the error-free transmission of packets; whereas, a host processor executes the "rare" case packet processing, which includes functions such as acknowledgment and retransmission of messages. The distinction is made primarily to simplify and thus execute faster the functions critical for improving the whole transmission or reception. For example, when transmitting a VMTP packet group, the processor updates the header delivery mask and appends a new buffer of data to the packet header between packets, queuing the new packet for the packet pipeline. However, one of the host processors is charged with forming the header for the first packet of the group because this task is more complex and requires close interactions with the user process and the operating system.

The on-board processor keeps a queue of TARs and RARs waiting for processing. The TARs for sending a single packet are directly queued for transmission. The others are added to a circular queue of TARs scanned periodically by a scheduler. The scheduling of TARs take into account the priority and the interpacket gaps timing constraints attached to each TAR. Typically, each ready TAR gets to send a packet before the next one can be sent from the same group of packets.

A Hashing algorithm is used to match the incoming packet's destination address to the locally acceptable addresses. The hash table entry contains the acceptable destination, local host number, local process identifier, and a pointer to a queue of waiting RARs. For the first packet in a group of packets, the packet's destination address is first used to locate the appropriate queue of RARs and then to match the appropriate RAR. The matched RAR is assigned to this group of packets and added to a list of recently matched RARs. This list is scanned first for the subsequent packets of this group of packets. In addition to filtering done by matching of RARs, it is also possible to filter packets based on a small number of (network-level or transport-level) source addresses from which all traffic is refused. The facility implements the "firewall" function which allows the NAB resources to be protected to some extent from failures or malicious behavior of a remote host processor or a user process. The selection of the appropriate source address to block on is done by a local host processor monitoring the traffic from the network.

The RARs and TARs remain indefinitely at the NAB until either the operation requested is completed or until a host requests them back by issuing a reset of the NAB. This "soft" reset operation allows a host to clear a destination's queue of waiting RARs or TARs, or all RARs and TARs at the NAB. Explicit timeouts per TAR or per RAR, based on the timeout values provided by a host, is an alternative way of handling the problem of unused RARs or TARs. The explicit requests to flush out the unused records minimizes the timer-related NAB processing, which is a non-trivial overhead especially at servers that may have many outstanding RARs and TARs. We expect the frequency of such flushes to be small, justifying the additional overhead of the "soft" resets.

4 Details about the Prototype

The prototype Network Adapter Board (NAB) has been designed using Motorola's MC68020 as the on-board processor, running at 16 Mhz clock rate; it uses about 200 hundred standard MSI and LSI components. The current version is designed for connecting two VMP multiprocessor system [3] with a 100 megabit/sec point-to-point connection.

The 256 kilobytes of buffer memory available through the random-access port is divided into two physical memory banks of equal size. Serial ports are connected to two 1024 bytes of serial-access memories, internal to the memory ICs, one per physical memory bank. Data transfers between a serial-access memory and the random-access memory take approximately 200 ns, and requires one to provide the row address on the address bus. A dual-port DRAM controller, an 8207, is used to arbitrate the use of the address bus by the packet pipeline, by the VMP bus, and by the MC68020.

A crossbar switch connects two serial access memories to transmit and receive pipeline, and read and write operations from the VMP bus, controlled either by host interface or by a block copier on the NAB. The serial access ports connected to the VMP system bus, which is a VME bus, allows block transfer (with the maximum block size of 1024 bytes) at 320 megabit/sec, using the serial mode of the VME bus. The serial access ports connected to packet pipelines transfers data at network rates up to 100 megabits per second. The other port, the random-access port, is connected only to the on-board processor. The serial-port controller stores buffer addresses provided by the on-board processor and controls the transfers between serial-access memories and the random-access memory.

The buffers are divided into two logical areas: one that contains packets received and one that contains data pulled in from the host memory. The mapping between the logical and the physical memory space is changed in order to move data from the host to the network without actually moving data. The crossbar switch, controlled by the logical-to-physical map setup by the on-board processor, directs the incoming data from the host or the network to the appropriate physical memory bank.

Transmit and receive packet pipelines are separate, allowing full-duplex communications over the network. These pipelines are 5 stages long, and use a 32-bit wide data path. The Receive pipeline also includes 4 words of FIFO storage, used for delaying the data stream until the decryption key is located. These pipelines work with the clock signals supplied by the Network Access Control (NAC) hardware.

The received packets are added to a queue of packets waiting for on-board processing. Addresses of empty buffers waiting for reception are buffered in a FIFO. The hardware reacts fast enough to accept back-to-back packets, with less than 500 nanoseconds gap, as long as buffers are available. On receiving an interrupt from the NAC, the receive buffer is linked to the receive queue by the on-board processor. Consecutive row addresses are used to receive packets larger than 1024 bytes.

The transmission and reception of packets over the network link are handled by the NAC hardware. The on-board processor sets up the pipeline to transmit the next packet. The packet is transmitted on a signal received from NAC, which implements the media access control protocol. The current version uses a point-to-point high-speed link, simplifying the media access control part of the NAC. The future versions of the NAC, designed to deal with multiaccess links such as the FDDI ring network [11], will have additional functions such as node address recognition and CRC checks. When a packet is fully transmitted, the on-board processor is interrupted. The on-board processor returns the packet buffer to the pool of empty buffers and sets the pipeline for the next transmit packet. On transmission of packets, a timer is used to help in implementing a fine-grained rate-based flow control policy, such as the one used in VMTP. The timer can be programmed to provide the inter-packet gap desired so that packets in a packet group do not overrun the receiver.

Host processors directly transfer data to the network adapter using a 1024-byte control register that is a part of the global memory address map. As soon as the control register is written by a host processor, its contents are transferred to a buffer that is queued for the attention of the on-board processor. The hardware is fast enough to permit back-to-back access to the control register without losing a host message, provided not all buffers are busy. The innermost loop of the software running at the NAB is examining these two receive queues and taking appropriate actions to receive packets, and to act on the RARs and TARs received from the VMP processor(s).

Large data segments are directly transferred between memory and network adapter by the NAB block copier. The NAB block copier contends for the VMP bus access like any other VMP host processor and transfers data between the buffer memory and the host memory using the serial transfer protocol of the VMP system bus. A VMP host is interrupted, by writing its host number into an interrupt register, to inform it that information is available in the control register.

5 Expected Performance

The two important cases to consider are delays for a small amount of data and for large data transfers. In addition, we also consider the reduction in processing load at servers, and host bus bandwidth. The expected performance is based on the timing information obtained from the prototype NAB design, and software processing time, estimated from the measurements of the VMTP implementation in the V distributed system. A lightly-loaded 100 megabit/sec FDDI ring is assumed for calculating the network delay.

5.1 Delay for Short Packets

Delay is the key issue in considering the performance for short packets. Figure 4 shows the estimated request-response time for a VMTP message transaction with no segment data, which results in a packet of minimum length, i.e., 64 bytes plus a network header.

The total delay includes the network latency, which is the waiting time to access the network, the packet transmission time, the bus transfer time, and the processing times at sender and receiver for both a request followed by a response packet. We assume the network latency of 100 microseconds, reflecting the average round-trip time in a moderate size lightly loaded FDDI ring. The packet size is 64 bytes of VMTP packet and network header is less than 16 bytes. The network delay, ignoring the small propagation delay, is the network latency plus the packet transmission time, 100 + (80 * 8 /100) microseconds, which is 106.4 microseconds. The estimated NAB processing

time at the sender is 50 microseconds, based on approximately 150 instructions needed to process the TAR, and to schedule the transmission. The estimated NAB processing time at the receiver, which includes the RAR matching time and the processing of a received packet, is 50 microseconds, assuming that an RAR is found. The packet processing times at a receiver and a sender, estimated from the actual CPU time observed for Send-Receive-Reply IPC in the V operating system, are 1576 and 1536 microseconds, respectively. These measurements are for a request followed by a response packet, both without a data segment between two SUN 3 workstations communicating over the 10-MB Ethernet. Using these values, we find that the total request-response time is approximately 3534 microseconds.

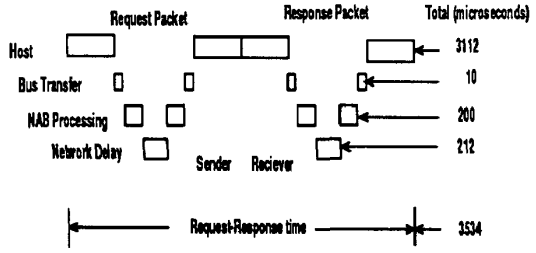


Figure 4: Request-response time for single short packet transfer

In Figure 4, we note that only about 12% of the total request-response time is spent in network adapters. The dominant factor is the host processing of the request and the response packet (about 88%). The network latency time and the NAB processing are each about 5.6% of the total delay. Thus, it would appear that further optimization of the NAB architecture for this case would yield marginal returns.

5.2 Throughput for Packet Groups

Figure 5 shows the total request-response time for transferring large amounts of data. To estimate this delay, we first consider the delay for host memory-to-memory data transfer. The request-response time is obtained by adding to this delay the host processing times at the sender and the receiver, plus the start-up time at the NAB for beginning a packet group transmission. Memory-to-memory transfer time is the network transmission time for a packet group, with the assumption that the packet processing required is less than one packet transmission time on the network. This is indeed the case in our design, once a packet group transmission begins. We neglect the small propagation delay of the medium, and we neglect the network latency, which in this case is small compared to the total data transmission time and could be made smaller by sending multiple packets in each access to the network. The packet size is 1024 bytes and the bus transfer rate is 320 megabit/sec. Thus, the buffering latency for the first packet is 25.6 microseconds. Finally, we conservatively estimate 50 microseconds as the start-up time at the NAB for beginning the packet group transmission.

From Figure 5, we calculate that the (expected) effective throughput for 16 kilobyte transfers is 44.3 megabit/sec. The memory-to-memory transfer rate is 94.5 megabit/sec. The buffering latency of the first packet and the start-up time are,

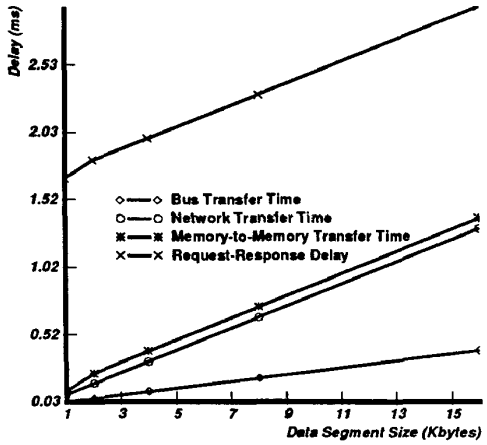


Figure 5: Request-response time for large data transfers

respectively, 0.8% and 1.7% of the total request-response time for reading 16 kilobytes. From Figures 4 and 5, we see that in the NAB architecture the Request-Response delay for both short and large data transfers, is dominated by the host processing time.

5.3 Impact on Host Resources

Server load as well as bus capacity used for communications is reduced by using host-network adapter interface, described in Section 2. In Figure 6, we have shown reduction of server load as the data segment size increases. The 100% load representes the CPU busy time as measured for VMTP implementation on SUN/2. The reduced server load allows a server to multiplex several concurrent transactions, providing an aggregate throughput approaching memory-to-memory transfer rates shown in Figure perfPG.

Reduced use of host bus bandwidth is illustrated in Figure 7. For each data segment size, the bus bandwidth used is compared with that necessary to checksum, encrypt, and packetize a data at host transferring data over the bus. Although using a cache to store data while performing these operations significantly reduces bus bandwidth, still better reduction is achieved by using the proposed interface.

5.4 Effect of Improving Processor Technology

The performance would improve if a powerful CPU is used both as the on-board processor and as a host processor. The faster CPU reduces the processing times at hosts and the start-up time for beginning the packet group transmission. Moreover, the reduced packet processing time at NAB, the cost of which is hidden in our design in the packet transmission time, allows one to use shorter packets with shorter transmission times; the shorter packets reduce the buffer latency time further decreasing the overall delay. The future workstations are also expected to use newer

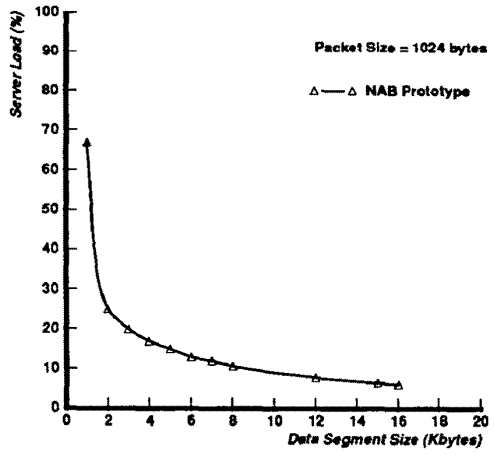


Figure 6: Reduction in Server Load due to NAB Interface

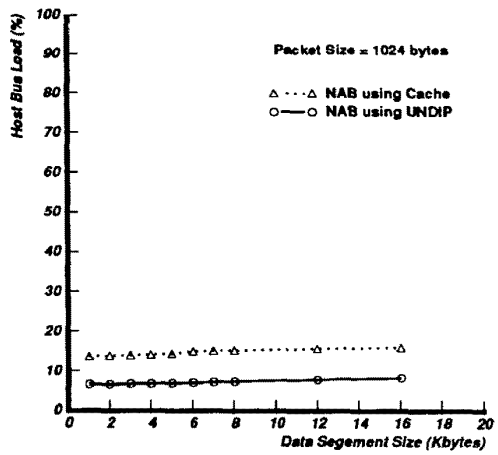


Figure 7: Relative use of Host Bus

and faster system bus designs such as the FutureBus [7], which can transfer data at 1.6 gigabit/sec. Taking all these factors in account we estimated Request-Response delay for large data transfers using 20-MIPS processors Figure 8. The packet size is 1024 bytes and the network speed is 100 megabit/sec. The processing times for processors with greater than 2 MIPS power are derived from our measurements on an MC68020, 2-MIPS machine, by reducing these figures by a factor proportional to the processor power relative to an MC68020.

In Figure 8, we note that the estimated throughput for 16 kilobyte transfers with the faster CPUs and a faster bus is 88.6 megabit/sec, and the memory-to-memory rate is within 1% of the network transfer rate (100 megabit/sec).

A well-balanced NAB architecture would match the NAB processor's capacity to the network data transfer rate, such that the time for processing a packet equals the time of receiving or transmitting a packet. In Figure 9, we show the expected throughput as the processing power increases, assuming that the system is well-balanced. The effective throughput is calculated for transfer of 16 kilobytes using maximum packet size of 1 kilobyte. The request-response throughput includes host processing time, which is 1575 microseconds at 2 MIPS, and which is proportionally reduced for a faster processor. The packet processing time within a packet group is estimated to take 75 microseconds for a 2-MIPS processor, based on approximately 150 instructions required for on board processing at the receiver. The processing time is reduced proportionally to obtain processing times (and thus transmission time) as the processing power increases. At 20-MIPS, as calculated from the the memory-to-memory transfer rate is approximately 1 gigabits/sec. The actual system performance is likely to be limited by the system bus capacity or the memory bandwidth; both of these are assumed here to be adequate.

6 Related Work

Sandy Frazer's Universal Receiver Protocol [12], developed at Bell Laboratories for the Datakit network, is one of the early attempts at developing a lightweight transport protocol to facilitate its hardware implementation. The URP recognizes that the bottleneck is at the receiving end, and thus concentrates on simplifying the receiver's state machine. It uses simple packet types and address fields. The packets are kept fairly small, reducing store-and-forward delay incurred at packet switches. Although useful in the environment it was developed, namely centralized-hub type networks, URP is not flexible enough to operate over subnets with widely differing characteristics.

Greg Chesson's Protocol Engine project [6] is a more recent example of the approach that depends on a lightweight transport protocol. The Protocol Engine is a specialized processor, to be implemented in one or two VLSI chips, that uses fast context switching to reduce packet processing overhead. In the P-Engine architecture, a specialized hardware block supports address recognition on reception of a packet. The other protocol functions such as retransmission, error control, and flow control are implemented in microcode. Although the protocol has more functionality than Sandy Frazer's URP protocol, on which it is based, it still lacks standard transport level features such as the support for encryption, multicasting, and process migration. Its error control scheme is a variation in the sliding window control scheme permitting only one gap in the sequence numbers of the received packets. Because of this variation, the performance is reduced in subnetworks that may reorder packets or split traffic across alternative routes to balance the load on the gateways.

Other efforts that have modified the processor design or changed microcode for efficient message communications include the Message-Driven Processor (MDP) by William Daley et al. [10], and Alfred Spector's PhD thesis work at Stanford University [18]. The MDP is a processing node for a message-passing concurrent computer. Spector microcoded an ALTO workstations to efficiently implement the small exchange of data over a network. Neither of these efforts were specifically focused

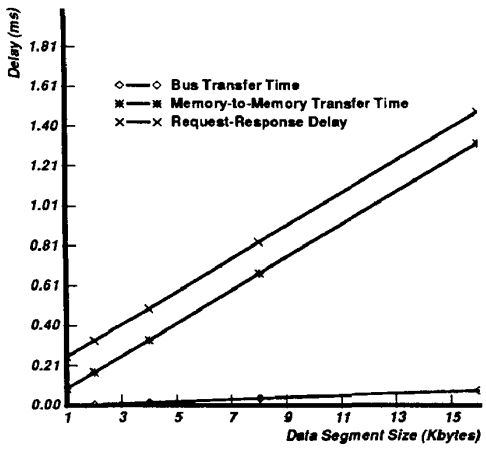


Figure 8: Performance improvements obtained with a 20-MIPS processor

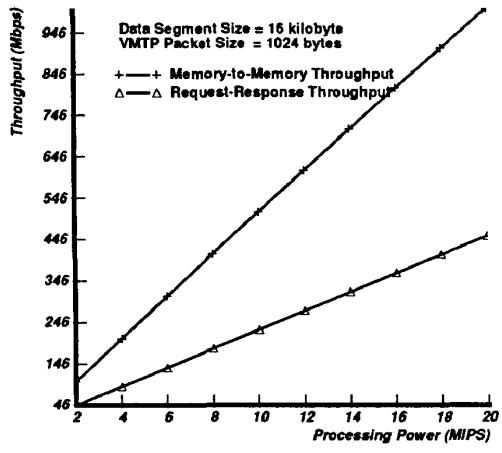


Figure 9: Throughput as Network Data Rate and NAB Processing Power increases.

on supporting transport-level functions.

Among the Ethernet devices and other network interfaces commercially available, one finds three basic models. We refer to these as the *memory model*, the *DMA model* and the *processor model*. In the memory model, a host processor reads packets from, and writes packets, to the network adapter as though it is simply a portion of memory, except for performing some operation to restart network packet transmission and reception after every packet. In the DMA model, the board accepts descriptors of packet buffers (possibly as scattered fragments of memory) and performs the packet transfer. In the processor model, the board copies data to and from the host memory plus provides some degree of processing capacity on-board. The VMP interface follows the memory model for small amounts of data. and the processor model for large amounts of data.

Several commercial produces use the memory model. The 3C400 [19] is an early example, providing only one transmit buffer and two receive buffers. A more recent and competitive design, the SUN Microsystem's Ethernet interface [23] provides 128 kilobytes of on-board memory plus network interface chip providing on-board scatter-gather DMA between the network and the on-board memory. One advantage cited for this design is that it allows direct DMA from a disk (for example) to the network interface, providing one bus transfer for data. However, this design assumes no software checksum and no encryption. Otherwise, additional memory references are incurred. Note also, either all file system buffers reside on the network interface board or else data must be transferred to main memory in any case. With the feasibility of dedicating multi-megabytes of memory to disk cache on a shared file server, the benefit of direct transfer to the network interface seems limited.

Several commercial products use the DMA model, including the DEC DEUNA/DEQNA boards [20]. This design forces a host processor to fully prepare a network packet for transmission in host memory, incurring at least 2 extra memory references and typically 4 if encryption is used. [6] Besides the copying cost, these interfaces interrupt on every packet transmission and reception, placing a significant overhead on the host processor(s). They also do not provide any "firewall" protection between the host and the network. In general, DMA network interfaces do not appear to have any advantages over the memory model.

A number of network interfaces in the processor model are available as commercial products. Examples include the Excelan 1020 [22] and the CMC ENP-100 [21]. In our experience, these interfaces are slower than commercial interfaces in the other models because of inadequate memory bandwidth on the board to serve the network transfers, hosts transfers and on-board processor accesses simultaneously. The on-board processor also introduces a delay between on-board reception of a packet and transfer to the network or host, depending on direction. To be fair, these products are aimed at providing network access for existing operating systems and architectures with minimal modification to the operating system. For instance, the CMC ENP board comes with a TCP implementation that makes it appear similar to a terminal multiplexor.

7 Concluding Remarks

High-performance communication is essential for distributed systems of the future. Network adapters are required to make the performance of future 100-1000 megabit networks available to the developing multiprocessor machines with minimal host and network overhead. We have described the design of the VMP network adapter and argued that it achieves these objectives, at least for 100-Mb networks.

We have presented a solution that incorporates several novel concepts in a host/adapter interface, a network adapter's internal architecture, and a communication protocol architecture. Al-

[6] The authors are not aware of any software implementation that performs the copy and checksum simultaneously. In the absence of this optimization, an additional memory reference is required.

though design is applicable to other multiprocessors and network protocols, the conventional independence between network adapter design, computer I/O system design, and transport protocol design seems incompatible with achieving the performance levels of interest here. In the following we summarize some of the important aspects of the design.

Host-Network Adapter Interface: The host-to-adapter interface treats small and large data transfers differently, handling the small transfers using a programmed I/O interface and the large data transfers with "intelligent" DMA. The programmed I/O interface minimizes delay for small packets. The DMA interface for large transfers minimizes cache and system bus traffic.

The adapter-to-host interface interrupts host on a data segment boundary and not for each packet transferred. For multi-packet transfers, this significantly reduces interrupt processing overhead, which is especially costly in multiprocessors systems, as one may need to trap data and code into the cache to handle interrupts.

The authorization model for reception allows a host to continue functioning even when being bombarded by packets from one or more hosts. This "firewall" function is essential for connecting hosts to a high-speed network.

Network Adapter Internal Architecture: On-board processing of checksumming, encryption, and packetization of data minimizes bus transfers to 1 per unit of transfer, namely a 32-bit word. This also avoids having to transfer data to the processor cache, which may improve the cache performance.

A packet pipeline, executing some key performance critical functions such as checksumming and encryption is used to increase throughput, particularly for large data transfers. The pipeline latency for short packet transfers is reduced by using few stages and a small unit for data transfers between the stages.

Contention-less memory accessing provided by a novel memory architecture based on Video RAMs reduces buffering latency and increases the packet processing rate. It allows processing of a packet by the on-board processor to proceed in parallel with the transfer of subsequent packets from the host to the buffer memory and from the network to the buffer memory.

Block copier hardware is used to transfer data at full blast between host memory and the adapter memory, thus reducing bus occupancy and buffering latency.

Communication Protocol Architecture: VMTP is designed specifically to provide high performance communications. Many features of VMTP are exploited in this design. The packet group concept is exploited by the host/adapter interface to minimize host interrupts. The packet group concept also simplifies the packet formation for packets in the same group. The NAB accepts a packet prototype of the first packet and forms headers of subsequent packets by making simple changes in it.

VMTP uses fixed size headers with space for small amounts of data and segment pointers for larger amounts, a feature exploited by the host/adapter interface to efficiently handle both small and large data transfers.

The VMTP checksum, which follows the data field, facilitates checksumming in hardware as the packet is transferred over the network, hiding its cost in a copying operation of transferring the data to the network.

In the prototype using M68020 processor, the throughput of a single data stream at the user level we have estimated falls short of network performance for networks in the 100 megabit range or higher. The aggregate throughput for concurrent transactions is closer to the 100 megabit range. Upgrading the prototype to use a high performance processor such as MIPS we estimated would push the throughput of a single transaction closer to the 100 megabit range. Further refinements in internal network adapter design and possibly host interface and transport protocols are required to push performance to a higher range. We see our current prototype and protocol design and implementation as but a first step to understanding how to design the next generation of network adapters.

8 Acknowledgments

This work was sponsored in part by the Defense Advanced Research Projects Agency under contract N00039-86-K-0431, by the Digital Equipment Corporation, the National Science Foundation Grant DCR-83-52048, AT&T Information Systems and Bell Northern Research.

References

[1] D. R. Cheriton. *VMTP: A Versatile Message Transaction Protocol*. Technical Report RFC 1045, Defense Advanced Research Projects Agency, February 1988.

[2] D. R. Cheriton and S.E. Deering. Host groups: A multicast Extension for Datagram Internetworks. In *9th Data Communication Symposium*, IEEE Computer Society and ACM SIGCOMM, September 1985.

[3] D. R. Cheriton, Gert Slavenberg, and Patrick Boyle. *Software-Controlled Caches in the VMP Multiprocessor*. Technical Report STAN-CS-86-1105, Computer Science Dept., Stanford University, 1986.

[4] D. R. Cheriton and C. L. Williamson. Network Measurement of the VMTP Request-Response Protocol in the V Distributed System. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 216–225, May 1987.

[5] D. R. Cheriton and W. Zwaenepoel. The Distributed V Kernel and its Performance for Diskless Workstations. In *Proceedings of 9th Symposium on Operating Systems Principles*, pages 129–140, ACM, October 1983.

[6] G. Chesson Protocol Engine project. Silicon Graphics Incorporated. A talk given at Stanford University, November 12, 1987.

[7] A special issue on computer bus standards. IEEE Micro. IEEE Computer Society, August 1984.

[8] *Connection Oriented Transport Protocol*. International Standards Organization, 1983. DP 8073.

[9] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milkiken, and T. Blakadar. Performance measurements on a 128-node butterfly parallel processor. In *Proceedings of the 1985 International Conference on Parallel Processing*, IEEE Computer Society, August 1985.

[10] W. Daley et al. Architecture of a Message-Driven Processor. In *Proceedings of the 14th Annual Intl. Symposium on Computer Architecture*, Pittsburg, PA. June 2-5, 1987.

[11] *Draft Proposed American National Standard, FDDI Token Ring Media Access Control - ANSC X3T9.5, 1986.*

[12] S. Fraser Universal Receiver Protocol. Internal memorandum. Bell Laboratories, Murray Hill.

[13] Z. Haas and D. R. Cheriton. A Case for Packet-Switching in High-Performance Wide-Area Networks. In *Proceedings of SIGCOMM 87 Workshop*, Stowe VT, Aug 11-13, 1987.

[14] J. L. Hennessy, N. Jouppi, F. Baskett and J. Gill. MIPS: A VLSI Processor Architecture. In *Proc. CMU Conference on VLSI Systems and Computations*, October 1981.

[15] H. Kanakia and F. Tobagi. Performance Measurements of a Data Link Protocol. In *International Conference on Communications*, IEEE, June 22-25 1986.

[16] F.A. Tobagi, F. Borgonova and L. Fratta Express-net: A High-performance Integrated-services Local Area Network. *IEEE J. on Selected Areas in Comm.*, Vol SAC-1, No. 5, Nov. 1983, pp. 898-912.

[17] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A Trace-driven Analysis of the UNIX 4.2 bsd File System. In *Proceedings of the Tenth Symposium on Operating Systems Principles*, pages 15–24, December 1985.

[18] A. Spector *Multiprocessing Architectures for Local Computer Networks*. Technical Report STAN-CS-81-874, Computer Science Dept., Stanford University, 1981.

[19] *3C400 Multibus Ethernet Controller: A Reference Manual.* 3Com Corporation, 1985.

[20] *DEUNA: User's Guide.* Digital Equipment Corp., 1983.

[21] *Ethernet Interface: User's Guide.* Communications Machinery Corp., 1985.

[22] *Excelan 1020: Network Interface - User's Guide.* Excelan Corp., Mountain View, Calif., 1985.

[23] *SUN-3 Architecture: A Sun Technical Report.* SUN Microsystems, Corp., 1985.