ATM and Fast Ethernet Network Interfaces for User-level Communication

Matt Welsh, Anindya Basu, and Thorsten von Eicken
{mdw,basu,tve}@cs.cornell.edu
Department of Computer Science
Cornell University, Ithaca, NY 14853, USA

Abstract

Fast Ethernet and ATM are two attractive network technologies for interconnecting workstation clusters for parallel and distributed computing. This paper compares network interfaces with and without programmable co-processors for the two types of networks using the U-Net communication architecture to provide low-latency and high-bandwidth communication. U-Net provides protected, user-level access to the network interface and offers application-level round-trip latencies as low as 60µsec over Fast Ethernet and 90µsec over ATM.

The design of the network interface and the underlying network fabric have a large bearing on the U-Net design and performance. Network interfaces with programmable co-processors can transfer data directly to and from user space while others require aid from the operating system kernel. The paper provides detailed performance analysis of U-Net for Fast Ethernet and ATM, including application-level performance on a set of Split-C parallel benchmarks. These results show that high-performance computing is possible on a network of PCs connected via Fast Ethernet.

1 Introduction

High-performance computing on clusters of workstations requires low-latency communication to efficiently implement parallel languages and distributed algorithms. Recent research [3, 8, 16] has demonstrated that direct application access to the network interface can provide both low-latency and high-bandwidth communication over commodity networks such as 155Mbps ATM and 100Base-TX Fast Ethernet. This paper presents two implementations of U-Net, a user-level network architecture employing off-the-shelf hardware, and compares their architectural properties and performance over ATM and Fast Ethernet.

U-Net circumvents the traditional UNIX networking architecture by providing applications with a simple mechanism to access the network device as directly as the underlying hardware permits. This shifts most of the protocol processing to user-level where it can often be specialized and better integrated into the application thus yielding higher performance. Protection is ensured through the virtual memory system and through kernel control of connection set-up and tear-down.

A previous implementation of U-Net over ATM[16] demonstrated that this architecture is able to efficiently support low-latency communication protocols such as Active Messages[17] for use as a workstation cluster interconnect for parallel computing. Split-C[5], a state-of-the-art parallel language, has been implemented using Active Messages over U-Net and, on a cluster of SPARCStations connected via ATM, shows performance comparable to MPPs such as the CM-5 and the Meiko CS-2. Recently a Fast Ethernet implementation [19] demonstrated that U-Net can be implemented efficiently over a network substrate other than ATM. U-Net over Fast Ethernet uses a substantially simpler network interface than the ATM implementation. This paper compares the two implementations and discusses the impact of the architectural differences on the software layers.

2 Motivation and Related Work

The U-Net architecture provides applications with direct access to the network interface without compromising protection. This allows protocol processing to be moved to user space and customized for specific applications. The intent is twofold:

- to reduce send and receive overheads for messaging so that the low latencies and high bandwidths required for cluster computing can be achieved, even with small message sizes; and
- to bring down the cost of workstation clusters through the use of inexpensive personal computers and a commodity interconnection network such as Fast Ethernet.

The U-Net architecture emphasizes low communication overheads because small messages are becoming increasingly important in a multitude of settings:

- in parallel programming languages where the granularity of remote accesses is often small and cannot easily be overlapped with unrelated computation, and which make abundant use of synchronization primitives (such as locks) where latency is critical;

- in object-oriented systems where objects may be distributed across the network and method invocations (involving small messages) may need to cross machine boundaries;
- for software fault-tolerance protocols (establishing consistent views of a distributed system among its members) which often require multiple rounds of small-message passing; and
- in network file systems in which the vast majority of messages are small (less than 200 bytes) in size.

2.1 The Case for Fast Ethernet

The initial implementation of U-Net over 140Mbps ATM (U-Net/ATM) demonstrated that low-latency communication for cluster computing is indeed possible using off-the-shelf hardware. Two important outstanding questions were whether the U-Net model is only feasible over connection-oriented networks such as ATM and whether the use of a programmable co-processor on the network adapter in the ATM implementation is a necessary part of the design.

The implementation of U-Net over Fast Ethernet (U-Net/FE) [19] explores the use of Fast Ethernet as an alternative to ATM. It shows that the U-Net design itself does not depend upon specific features of ATM networks or on the use of a programmable co-processor on the network interface. Fast Ethernet has a number of technical and cost advantages over ATM. First, Fast Ethernet is a mature technology with well-known standards (the basic design remains that of the original 10Mbps Ethernet system) and products are widely available. Second, network adapters, cabling, hubs, and switches for 100Mbps Fast Ethernet are significantly less expensive than their ATM counterparts. As an example, high-end ATM network interfaces generally cost five to ten times more than high-end Fast Ethernet adapters with similar features.

The lower cost of Fast Ethernet is primarily due to two factors: volume and simplicity. The seamless integration of Fast Ethernet into legacy networks creates a high-volume market and makes it far more attractive than ATM, which can be difficult to integrate into existing networks. In addition, the cell segmentation and reassembly process required in ATM is more costly to implement than the simpler DMA block transfers which suffice for Fast Ethernet.

2.2 Related Work

User-level networking issues have been studied in a number of recent projects. Several of these models propose to introduce special-purpose networking hardware.

Thekkath[14] proposes to separate the control and data flow of network access using a shared-memory model; remote-memory operations are implemented as unused opcodes in the MIPS instruction set.

The Illinois Fast Messages[12] achieve high performance on a Myrinet network using communication primitives similar to Active Messages. The network interface is accessed directly from user-space but does not provide support for simultaneous use by multiple applications.

The HP Hamlyn[20] network architecture also implements a user-level communication model similar to Active Messages but uses a custom network interface where message sends and receives are implemented in hardware.

Shrimp[3] allows processes to connect virtual memory pages on two nodes through the use of custom network interfaces; memory accesses to such pages on one side are automatically mirrored on the other side.

The ParaStation[18] system obtains small-message (4-byte) send and receive processor overheads of about 2.5µsec using specialized hardware and user-level unprotected access to the network interface. The Beowulf[2] project has constructed a workstation cluster consisting of Pentium systems connected via Fast Ethernet. Each system consists of two Fast Ethernet controllers operating in a round-robin fashion to double the aggregate bandwidth per node. This project employs the same network hardware and operating system as U-Net/FE, however, all network access is through the kernel sockets interface.

Similarly, the Berkeley Network-of-Workstations[1] project aims to form large-scale distributed and parallel computing systems out of off-the-shelf components, connected via FDDI or Myrinet.

3 U-Net communication architecture

The U-Net architecture virtualizes the network interface in such a way that a combination of operating system and hardware mechanisms can provide every application the illusion of owning the interface to the network. Depending on the sophistication of the actual hardware, the U-Net components manipulated by a process may correspond to real hardware in the NI, to software data structures that are interpreted by the OS, or to a combination of the two. The role of U-Net is limited to multiplexing the actual NI among all processes accessing the network and enforcing protection boundaries. In particular, an application has control over both the contents of each message and the management of send and receive resources.

3.1 Sending and receiving messages

U-Net is composed of three main building blocks, shown in Figure 1: *endpoints* serve as an application's handle into the network and contain a *buffer area* to hold message data, and *message queues* to hold descriptors for

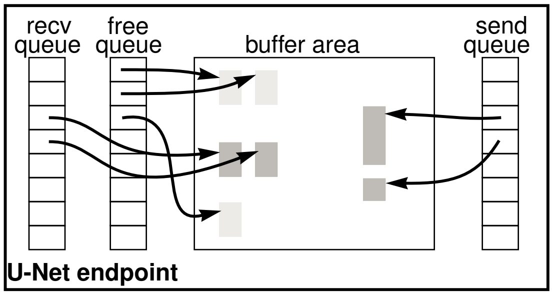


Figure 1: U-Net building blocks. *Endpoints* serve as an application's handle into the network, *buffer areas* are regions of memory that hold message data, and message queues *(send/recv/free queues)* hold descriptors for messages that are to be sent or that have been received.

messages that are to be sent or that have been received. Each process that wishes to access the network first creates one or more endpoints. Communication between endpoints occurs through *communication channels* — a communication channel is associated with a pair of endpoints and a channel identifier (usually a small integer) that is assigned to it at the time of creation.

Communication channel identifiers, in conjunction with *message tags*, are used to uniquely identify the source and destination of an individual message. The exact form of a message tag depends on the network substrate — for example, for ATM networks, virtual channel identifiers (VCIs) may be used as message tags. An application registers these tags with U-Net when it creates a communication channel — an operating system service is needed to assist the application in determining the correct tag to use based on a specification of the destination process and the route between the two nodes.[1]

To send a message, a user process composes the data in the endpoint buffer area and pushes a descriptor for the message onto the send queue. The network interface then transmits the message after marking it with the appropriate message tag.

Incoming messages are demultiplexed based on the message tag. The data is then transferred into one or more free buffers (in the buffer area of the recipient endpoint) provided by the application and a message descriptor with pointers to the buffers is pushed onto the appropriate receive queue. As an optimization for small messages—which are used heavily as control messages in protocol implementations—a receive queue descriptor may hold an entire small message (instead of buffer pointers). This avoids buffer management overheads and can improve the round-trip latency substantially. The size of these small messages is implementation-dependent and typically reflects the properties of the underlying network.

The receive model supported by U-Net is either polling or event-driven: the process can periodically check the status of the receive queue, it can block waiting for the next message to arrive (using a UNIX *select* call), or it can register a signal handler with U-Net which is invoked when the receive queue becomes non-empty. In order to amortize the cost of an upcall over the reception of several messages U-Net allows all messages pending in the receive queue to be consumed in a single upcall.

The management of the transmit and receive buffers is entirely up to the application: the U-Net architecture does not place constraints on the size or number of buffers nor on the allocation policy used. The application provides receive buffers explicitly to the NI via the free queue but it cannot control the order in which these buffers are filled with incoming data.

4 Comparing the U-Net Implementations

The two U-Net implementations compared in this paper differ substantially due to the significant architectural differences between the two networking technologies. The Fast Ethernet version is implemented entirely within the kernel while the ATM version uses custom firmware in the network interface co-processor. The main differences are the following:

- The size and granularity of network data units: ATM packets in the AAL5 format must be segmented into 48-byte cells, and the maximum packet size is 65KBytes. Ethernet frames, on the other hand, can hold between 46 to 1500 bytes of payload each; larger packets must be fragmented.
- Multiplexing: ATM allows data to be multiplexed on a link at the relatively fine granularity of cells, while in Ethernet the transmission of a packet of up to 1500 bytes is indivisible.
- Connection-oriented versus connection-less: ATM is fundamentally a connection-oriented network, where the network assigns a VCI to each end-to-end (application-to-application) connection. Ethernet, however, is primarily packet-switched and no connection set-up is necessary. Moreover, the Ethernet source and destination addresses identify only a particular network interface, not an individual application endpoint.
- Shared versus switched medium: Ethernet has tradition-

1. The operating system service will assist in route discovery, switch-path setup and other (signalling) tasks that are specific to the network technology used. The service will also perform the necessary authentication and authorization checks to ensure that the application is allowed access to the specific network resources and that there are no conflicts with other applications. After the path to the peer has been determined and the request has passed the security constraints the resulting tag will be registered with U-Net such that the latter can perform its message multiplexing/demultiplexing function. A channel identifier is returned to the requesting application to identify the communication channel to the destination.

ally been a shared-medium network where all stations compete for use of the wire, using exponential backoff algorithms for retransmission in case of collision. ATM is switched in the sense that every station has a point-to-point connection to a local router. Thus, the use of Fast Ethernet for high-performance communication raises the concern that contention for the shared medium might degrade performance as more hosts are added to the same network. However, Fast Ethernet switches are available (typically at lower cost than comparable ATM switches) and offer each station a "private" link to the network. Such a "private" link can be a full-duplex link which allows a host to simultaneously send and receive messages (as opposed to a shared half-duplex link) and thus doubles the aggregate network bandwidth.

4.1 Experimental Set-up

The experimental configuration consists of a cluster of Pentium workstations, running the Linux operating system, connected via Fast Ethernet and ATM. The network interface for the ATM interconnect is the Fore Systems PCA-200 that includes an on-board processor which performs the segmentation and reassembly of packets as well as transfers data to/from host memory using DMA. The PCA-200 consists of a 25Mhz Intel i960 processor, 256Kbytes of memory, a DMA-capable PCI-bus interface, a simple FIFO interface to the ATM fiber, and an AAL5 CRC generator/checker. The i960 processor is controlled by firmware which is downloaded into the on-board RAM by the host. The host processor can map the PCA-200 memory into its address space in order to communicate with the i960 during operation. The U-Net implementation on this interface uses custom firmware to implement the U-Net architecture directly on the PCA-200. The ATM switch used is a Fore Systems ASX-200 which forwards cells in about 7µs.

The network interface for the Fast Ethernet interconnect uses the DECchip 21140 Fast Ethernet controller. The DC21140 is a PCI bus master capable of transferring complete frames to and from host memory via DMA. It includes a few on-chip control and status registers, a DMA engine, and a 32-bit Ethernet CRC generator/checker. The board maintains circular send and receive rings, containing descriptors which point to buffers for data transmission and reception in host memory. The design of the DC21140 assumes that a single operating system agent will multiplex access to the hardware. Therefore, coordination with the host OS is necessary to allow protected access to multiple user applications.

Several Fast Ethernet hubs and switches were used to benchmark the network interface. A Bay Networks 100BaseTX hub, a Bay Networks 28115 16-port switch and a Cabletron FN100 8-port switch were individually employed.

4.2 ATM Network Interface Operation

The U-Net implementation for the PCA-200 uses custom firmware to implement U-Net directly and is largely identical to that of the Sbus-based SBA-200 described in [16]. The firmware allows multiple user processes to concurrently communicate with the on-board i960 which maintains a data structure that contains protection information for all open endpoints. Buffer areas are pinned to physical memory and mapped into the i960's DMA space allowing direct transfers between user space and the physical network queues. Receive queues are allocated in main memory so that the host can poll them without crossing the I/O bus, while send and free queues are placed in PCA-200 memory and mapped into user-space so that the i960 can poll these queues without DMA transfers.

4.2.1 Endpoint and Channel Creation

Creation of user endpoints and communication channels is managed by the operating system. Applications use the system call interface to the device driver to create endpoints and channels. The device driver validates these system call requests and passes the appropriate commands to the i960 using a special command queue. This is necessary to enforce protection boundaries between processes and to properly manage system resources. Communication channels are associated with a pair of endpoints identified by a virtual channel identifier (VCI) pair. The VCIs are used as message tags to route outgoing messages and demultiplex incoming messages. The buffer areas and message queues for distinct endpoints are disjoint and are only mapped to the address space of the process that creates the endpoint.

4.2.2 Transmission

In order to send a message, the host stores the U-Net send descriptor into the i960-resident transmit queue using a double-word store. The i960 periodically polls each transmit queue for new entries — endpoints with recent activity are polled more frequently given that they are most likely to correspond to a running process. Once the i960 finds a new transmit descriptor, it initiates DMA transfers from the user-space buffer(s) to the network output FIFO on the SBA-200/PCA-200. For large messages, the DMA occurs in 32-byte bursts on the Sbus and 96-byte bursts on the PCI bus.

4.2.3 Reception

The i960 periodically polls the network input FIFO and processes incoming cells one at a time. For each cell, it uses the VCI to index into a table which indicates the appropriate destination endpoint. When the first cell of a message arrives, the i960 allocates a buffer from the end-

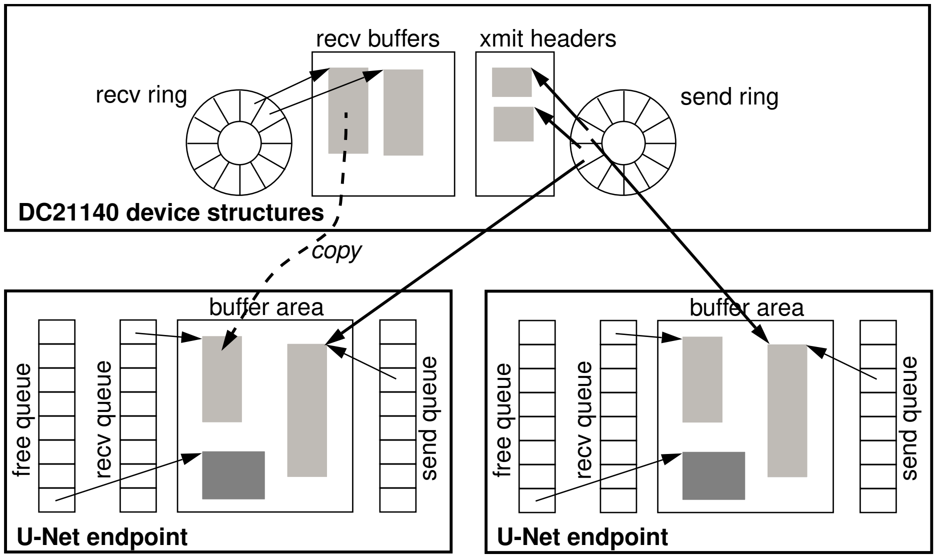


Figure 2: U-Net/FE endpoint and device data structures.

point's free queue and transfers the cell into the buffer. Additional cells are appended one at a time to the buffer. When the last cell of a message is received, the i960 checks the CRC (which is accumulated in hardware) and places a descriptor into the endpoint's receive queue.

Receiving single-cell messages is special-cased to improve the round-trip latency for small messages — such messages are directly transferred into the next empty receive queue entry (which is large enough to hold the entire message) and thus avoids the overheads of buffer allocation and extra DMA for the buffer pointers.

4.3 Fast Ethernet Network Interface Operation

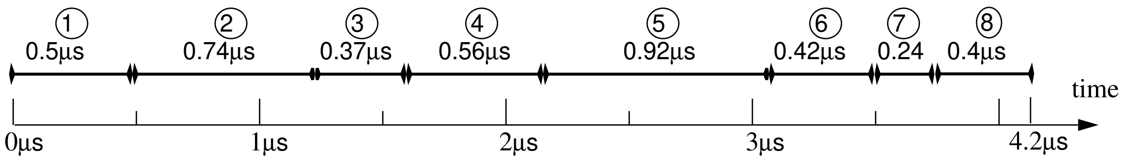
The DC21140 PCI Fast Ethernet controller used in the U-Net implementation provides a straightforward interface based on transmit and receive buffer descriptor rings. This interface was designed for traditional in-kernel networking layers in which the network interface is controlled by a single agent on the host. In order to multiplex the network interface among user processes, the U-Net implementation must be placed in the kernel which differs significantly from U-Net/ATM.

The in-kernel implementation of U-Net is best described as a protected co-routine available to user processes. User processes can issue a fast trap into kernel space which services the U-Net transmit queue in a manner similar to the i960 in the ATM implementation. When network packets arrive, an interrupt is generated by the DC21140 which transfers control to the in-kernel U-Net routines for message reception. In this sense a portion of main processor time is allocated to servicing U-Net requests by user processes, while in U-Net/ATM a dedicated co-processor is employed for this task.

The DC21140's transmit and receive descriptor rings are stored in host memory: each descriptor contains pointers to up to two buffers (also in host memory), a length field, and flags. Multiple descriptors can be chained to form a PDU out of an arbitrary number of buffers. These descriptor rings must be shared among all U-Net/FE endpoints and are therefore distinct from the U-Net transmit and receive queues stored in the communication segment. Figure 2 shows the various rings, queues and buffer areas used in the U-Net/FE design.

4.3.1 Endpoint and Channel Creation

A communication channel in the U-Net/FE architecture is associated with a pair of endpoints, each of which is identified by a combination of a 48-bit Ethernet MAC address and a one-byte U-Net port ID. The MAC address and port ID combinations are used as message tags in the U-Net/FE architecture. A communication channel is created by issuing a system call to the U-Net device driver and specifying the two sets of Ethernet MAC addresses and port IDs. The Ethernet MAC address is used to route outgoing messages to the correct interface on the network while the port ID is used to demultiplex incoming messages to a particular endpoint. The operating system registers the requested addresses and returns a channel identifier to the application. The channel identifier is subsequently used by the application to specify a particular end-to-end connection when pushing entries onto the U-Net send queue. Similarly, the operating system uses the incoming channel identifier when placing new entries on the receive queue for the application.



1. trap entry overhead
2. check U-Net send parameters
3. Ethernet header set-up
4. device send ring descriptor set-up
5. issue poll demand to DC21140 to start TX process
6. free send ring descriptor of previous message
7. free U-Net send queue entry of previous message
8. return from trap

Figure 3: Fast Ethernet transmission timeline for a 40 byte message (60 bytes with the Ethernet and U-Net headers)

4.3.2 Packet Transmission

To send a message, an application constructs the message in the endpoint buffer area and pushes an entry onto the U-Net send queue. The application issues a fast trap to the kernel where the U-Net driver services the user's send queue. This is implemented as an x86 trap gate into kernel space, requiring under 1µs for a null trap on a 120 Mhz Pentium system. This form of trap does not incur the overhead of a complete system call, and the operating system scheduler is not invoked upon return.

The kernel service routine traverses the U-Net send queue and, for each entry, pushes corresponding descriptors onto the DC21140 send ring. Each ring descriptor contains pointers to two buffers: the first is an in-kernel buffer with the Ethernet header and packet length field, and the second is the user buffer containing the data (for multi-buffer user messages additional descriptors are used). By pointing directly to the U-Net buffer area, a copy is avoided and the DC21140 can transmit data directly from user-space. After all descriptors have been pushed onto the device transmit ring, the in-kernel service routine issues a transmit poll demand to the DC21140 which initiates the actual transmission.

4.3.3 Packet Reception

Upon packet reception the DC21140 transfers the data into buffers in host memory pointed to by a device receive ring analogous to the transmit ring. These are fixed buffers allocated by the device driver and are used in FIFO order by the DC21140. The DC21140 generates an interrupt, the kernel interrupt routine determines the destination endpoint and channel identifier from the U-Net port number contained in the Ethernet header, copies the data into the appropriate U-Net buffer area and enqueues an entry in the user receive queue. As an optimization, small messages (under 64 bytes) are copied directly into the U-Net receive descriptor itself.

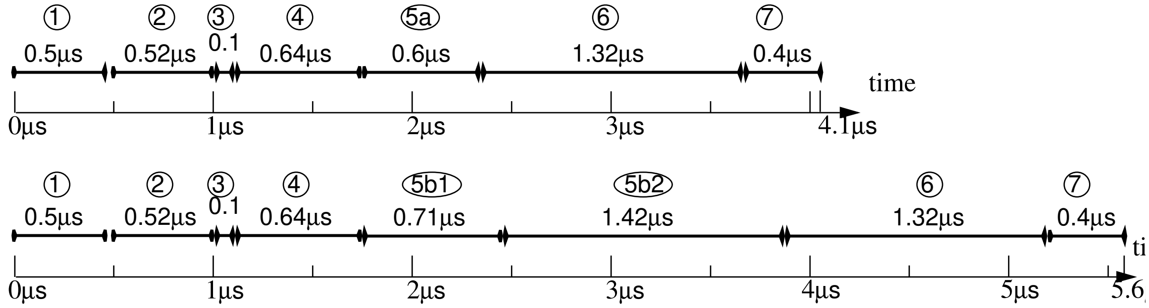
4.4 Performance and Discussion

Although U-Net cannot be implemented directly on the Fast Ethernet interface itself due to the lack of a programmable co-processor, the kernel trap and interrupt handler timings demonstrate that the U-Net model is well-suited to a low-overhead in-kernel implementation. The processor overhead for sending a message, independent of size, is approximately 4.2µs. While a co-processor could off-load the Pentium, few (if any) could perform the necessary queue management functions in less time. In addition, allowing the U-Net queue management to take place on the host processor is beneficial as host memory access from the Pentium does not incur overheads for bus transfers. In contrast, network interface co-processors must cross the system bus to manage queues in host memory.

4.4.1 Transmission and reception timings

The timeline for transmission of a 40-byte message on U-Net/FE is shown in Figure 3. The timings were obtained using the Pentium cycle counters and using repeated executions of parts of the trap code. The timeline corresponds to the transmission of a 60-byte Ethernet frame including the U-Net and Ethernet headers. A timing analysis of the U-Net trap code shows that the processor overhead required to push a message into the network is approximately 4.2µs of which about 20% are consumed by the trap overhead. In contrast, the processor overhead for sending a 40-byte message on U-Net/ATM is about 1.5µsec while the i960 overhead is about 10µsec.

Figure 4 shows the timeline for reception of 40- and 100-byte messages by U-Net/FE. The short message optimization is effective as 15% overhead is saved by not allocating a separate receive buffer. For messages of more than 64 bytes the copy time increases by 1.42µs for every additional 100 bytes. The latency between frame data arriving in memory and the invocation of the interrupt handler is roughly 2µs and the major cost of the receive interrupt handler is the memory copy required to place incoming data into the appropriate user buffer area. The Pentium memory-copy speed is about 70Mbytes/sec and the DMA



1. interrupt handler entry
2. poll device recv ring
3. demux to correct endpoint
4. alloc+init U-Net recv descr
5a. copy 40 byte message
5b1. allocate U-Net recv buffer
5b2. copy 100 byte message
6. bump device recv ring
7. return from interrupt

Figure 4: Fast Ethernet reception timeline for a 40-byte and a 100-byte message. With the Ethernet and U-Net headers these correspond to 60 and 116 byte frames.

of incoming frames from the DC21140 is pipelined with the copy within the interrupt handler. The primary disadvantage of the additional copy is processor utilization during message receive. In comparison, the receive overhead for the i960 for a 40-byte message (which does not require the allocation of a receive buffer) is approximately 13µs.

4.4.2 Bandwidth and Round-trip Latency

Figure 5 depicts the application-to-application message round-trip time as a function of message size for U-Net/FE on the DC21140 and U-Net/ATM on the FORE PCA-200. Message sizes range from 0 to 1498 bytes, the maximum PDU supported by U-Net/FE; although the PDU limit on ATM is 64Kbytes, corresponding to the MTU of AAL5. Three Fast Ethernet round-trip times are shown: with a broadcast hub, with a Bay Networks 28115 16-port switch, and with a Cabletron FastNet100 8-port switch. The round-trip time for a 40-byte message over Fast Ethernet ranges from 57µsec (hub) to 91µsec (FN100), while over ATM it is 89µsec[2]. This corresponds to a single-cell send and receive which is optimized for ATM. The inset depicts round-trip times for small messages (between 0 and 128 bytes).

The increase in latency over Fast Ethernet is linear with a cost of about 25µsec per 100 bytes; over ATM, the increase is about 17µsec per 100 bytes. This can be attributed in part to the higher serialization delay over 100Mbps Fast Ethernet as opposed to 155Mbps ATM. Longer messages (i.e. those that are larger than a single cell) on ATM start at 130µsec for 44 bytes and increase to 351µsec for 1500 bytes. This sharp rise can be attributed to the fact that both transmit and receive on U-Net/ATM are optimized for single cell sends and receives, in particular, a single cell receive does not involve the additional cost of receive buffer allocation. Similar behavior (although not as pronounced) is shown by the U-Net/FE graphs in the neighborhood of 64 bytes, which is the threshold for the small-message optimization.

2. U-Net over ATM on 140Mbps TAXI achieves 65µs round-trip latency [16]; the additional overhead here is incurred due to OC-3c SONET framing.

Figure 6 depicts bandwidth in Mbits/sec over U-Net for Fast Ethernet and ATM with message sizes ranging from 0 to 1498 bytes. For messages as small as 1Kbyte the bandwidth approaches the peak of about 97Mbps (taking into

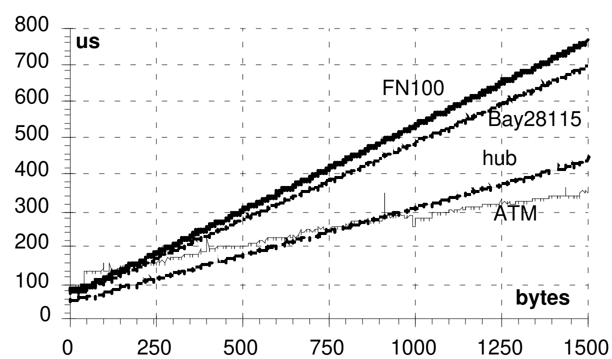
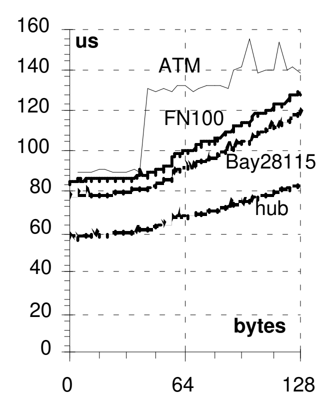
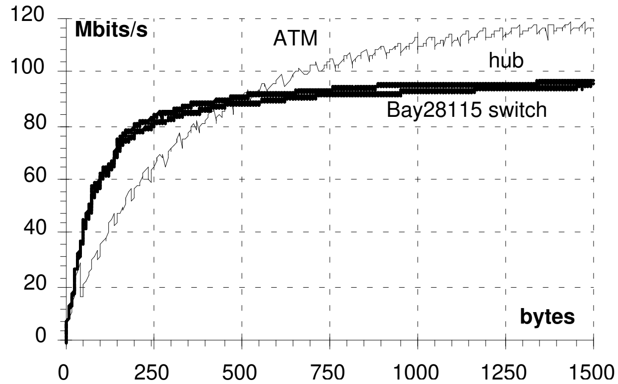


Figure 5: Round-trip message latency vs. message size for Fast Ethernet and ATM. The graph on the right magnifies the small-message latency measurements. Fast Ethernet measurements were taken using a broadcast hub, a Bay Networks 28115 16-port switch, and a Cabletron FastNet100 8-port switch The ATM measurements us a Fore ASX-200 switch.





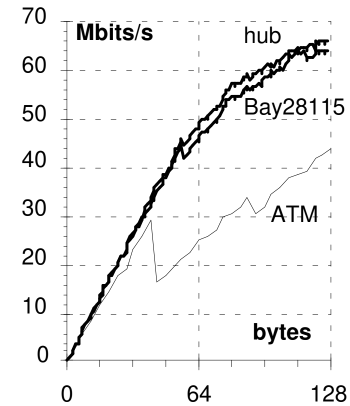


Figure 6: Bandwidth vs. message size for Fast Ethernet and ATM. Fast Ethernet saturates at around 96Mbps while ATM reaches 118Mbps (this measurement uses a 120Mbps link in the network). The jagged ATM measurement is due to the segmentation into fixed-size cells.

account Ethernet frame overhead) for Fast Ethernet. Due to SONET framing and cell-header overhead the maximum bandwidth of the ATM link is not 155Mbps, but rather 138Mbps. The maximum bandwidth here is 120 Mbps, which represents the maximum achievable bandwidth for the 140Mbps TAXI link used as the receiving end for this benchmark.

4.4.3 Discussion

It is evident from the above performance figures that the nature of the network interface has significant effect on the performance. The U-Net/FE architecture, while simple, sacrifices overlap of communication and computation for lower message latencies. This is clear from the send overheads for a 40-byte message: while the total send overhead for U-Net/FE is 5.4µs, the total send overhead for U-Net/ATM is approximately 11.5µs, almost double. However, the processor overheads are dramatically different in the two cases: the U-Net/FE architecture shows an overhead of 4.2µs while that for U-Net/ATM is 1.5µs.

Communication patterns involving a great deal of synchronization are suited to U-Net/FE as latency is lower, although this comes at the cost of trapping to the kernel for send and receive. In contrast, communication over U-Net/ATM incurs a very low processor overhead at the cost of off-loading to a slow network interface co-processor. The U-Net/ATM architecture is suitable for applications which pipeline many message transmissions and synchronize rarely, in particular applications requiring high bandwidth. These observations are further supported by application benchmark results in the next section.

Another issue to be addressed is scalability. The use of Ethernet MAC addresses and port IDs to address endpoints does not allow messages to traverse multiple switches or IP routers. One solution would be to use a simple IPv4 encapsulation for U-Net messages; however, this would add considerable communication overhead. U-Net/ATM does not suffer this problem as ATM virtual circuits are established network-wide.

5 Parallel Algorithm Benchmarks

A set of parallel algorithm benchmarks written in the Split-C [5] language have been employed to compare high-level application performance of the two U-Net implementations. The Split-C language allows processes to transfer data through the use of *global pointers* — a virtual address coupled with a process identifier. Dereferencing a global pointer allows a process to read or write data in the address space of other nodes cooperating in the parallel application. Split-C is implemented over Active Messages [17], a low-cost RPC mechanism, providing flow control and reliable transfer, which has been implemented over U-Net [16].

The Fast Ethernet experimental platform consists of a cluster of one 90 MHz and seven 120-MHz Pentium workstations running Linux 1.3.71 and connected by a Bay Networks 28115 16-port switch to a 100Mbps Fast Ethernet network, while the ATM experimental platform consists of a cluster of 4 SPARCStation 20s and 4 SPARCStation 10s running SunOS 4.1.3 and connected by a Fore ASX-200 switch to a 140 Mbps ATM network[3], using the FORE Systems SBA-200 SBus ATM adaptor. The SBA-200 implementation of U-Net is largely identical to that for the PCA-200 described here.

5.1 Benchmark Description

The Split-C benchmark suite consists of five programs: a blocked matrix multiply, a sample sort optimized for small and large message transfers, and a radix sort optimized for small and large message transfers. The performance of this benchmark suite on a variety of multiprocessors is presented in [5].

The matrix multiply application was run twice, once using matrices of 8 by 8 blocks with 128 by 128 double floats in each block, and once using 16 by 16 blocks with 16 by 16 double floats in each block. The main loop in the

3. The use of SPARCstations rather than Pentiums connected via ATM was necessitated by lack of available PCA-200 interfaces. As demonstrated by the benchmarks the computational capabilities of these machines are very comparable.

matrix multiply algorithm repeatedly fetches a block from each of the two matrices to be multiplied, performs the multiplication, and stores the result locally.

Both the radix and sample sort benchmarks sort an array of 32-bit integers over all nodes. Each node has 512K keys with an arbitrary distribution. The radix sort uses alternating phases of local sort and key distribution involving irregular all-to-all communication. The algorithm performs a fixed number of passes over the keys, one for every digit in the radix. Each pass consists of three steps: first, every processor computes a local histogram based on its set of local keys; second, a global histogram is computed from the local histograms to determine the rank of each key in the sorted array; and finally, every processor sends each of its local keys to the appropriate processor based on the key's rank in the sorted array. In the version optimized for small messages, each processor transfers two keys at a time in the last step of each pass. In the version optimized for large messages, each processor sends one message containing all relevant keys to every other processor during the last step of each pass.

Instead of alternating computation and communication phases, the sample sort algorithm uses a single key distribution phase. The algorithm selects a fixed number of samples from keys on each node, sorts all samples from all nodes on a single processor, and selects splitters to determine which range of key values should be used on each node. The splitters are broadcast to all nodes. The main communication phase consists of sending each key to the appropriate node based on splitter values. Finally, each node sorts its values locally. The small-message version of the algorithm sends two values per message while the large-message version transmits a single bulk message.

5.2 Performance

The absolute execution times for benchmark runs on two, four and eight nodes of both the Pentium Fast Ethernet cluster and the SparcStation ATM cluster are shown in Table 1. Execution times normalized to the 2-node SparcStation ATM cluster are shown in Figure 7.. All benchmarks have been instrumented to measure communication and computation time separately

Benchmark	ATM 2 nodes	FE 2 nodes	ATM 4 nodes	FE 4 nodes	ATM 8 nodes	FE 8 nodes
mm 128x128	56.59	117.00	33.31	54.68	29.04	48.52
mm 16x16	1.26	1.67	0.90	1.04	0.56	0.67
ssortsm512K	1.08	0.63	1.26	0.78	1.69	1.08
ssortlg512K	2.03	2.06	2.11	2.55	2.93	3.22
rsortsm512K	48.13	44.61	76.64	63.79	88.23	90.50
rsortlg512K	4.73	5.35	5.01	6.30	7.15	7.54

Table 1: Execution Times for Split-C Benchmarks (in seconds)

The overall results demonstrate that performance on both U-Net implementations scales well when the number of processors is increased. Table 2 shows the speedup from 2 to 8 nodes for both U-Net/FE and U-Net/ATM. In the case of matrix multiplication, the matrix size is kept constant for all clusters as demonstrated by the corresponding reduction in execution time. In the case of the radix and sample sorts, the number of keys per processor is kept constant, explaining the increased total execution time from 2 to 8 nodes.

Benchmark	ATM	FE
mm 128x128	1.9	2.4
mm 16x16	2.2	2.5
ssortsm512K	2.5	2.4
ssortlg512K	2.9	2.5
rsortsm512K	2.2	2.0
rsortlg512K	2.7	2.9

Table 2: Speedup for ATM and FE clusters (from 2 to 8 nodes)

Two factors explain the matrix multiply performance advantage over ATM. First, large messages are used which benefit from higher bandwidth. Second, SPARC floating-point operations outperform those of the Pentium. The small-message versions of the sample and radix sort benchmarks are dominated by network time, and Fast Ethernet outperforms ATM due to lower overhead. In addition, Pentium integer operations outperform those of the SPARC. Increased synchronization overheads as the number of processors is increased accounts for the additional communication time on 4 and 8 nodes. ATM outperforms Fast Ethernet for the large-message versions of the sample and radix sort benchmarks, primarily due to increased network bandwidth. We cannot account for the anomalous increase in computation time as the number of processors increase for sample sort.

In summary, the Fast Ethernet cluster demonstrates higher performance when low message latencies and integer operations dominate; the ATM cluster demonstrates higher performance when higher bandwidth and floating-point performance are required.

6 Summary and Conclusions

U-Net has been presented as an efficient user-level communication architecture over Fast Ethernet, with performance rivaling that of 155 Mbps ATM. We have shown that U-Net can be extended to networks other than ATM, as well as to network interfaces without a programmable co-processor, where the OS kernel is required to intervene in the critical path.

A detailed timing analysis of the U-Net/FE trap code shows that processor overhead for transmit is small, while

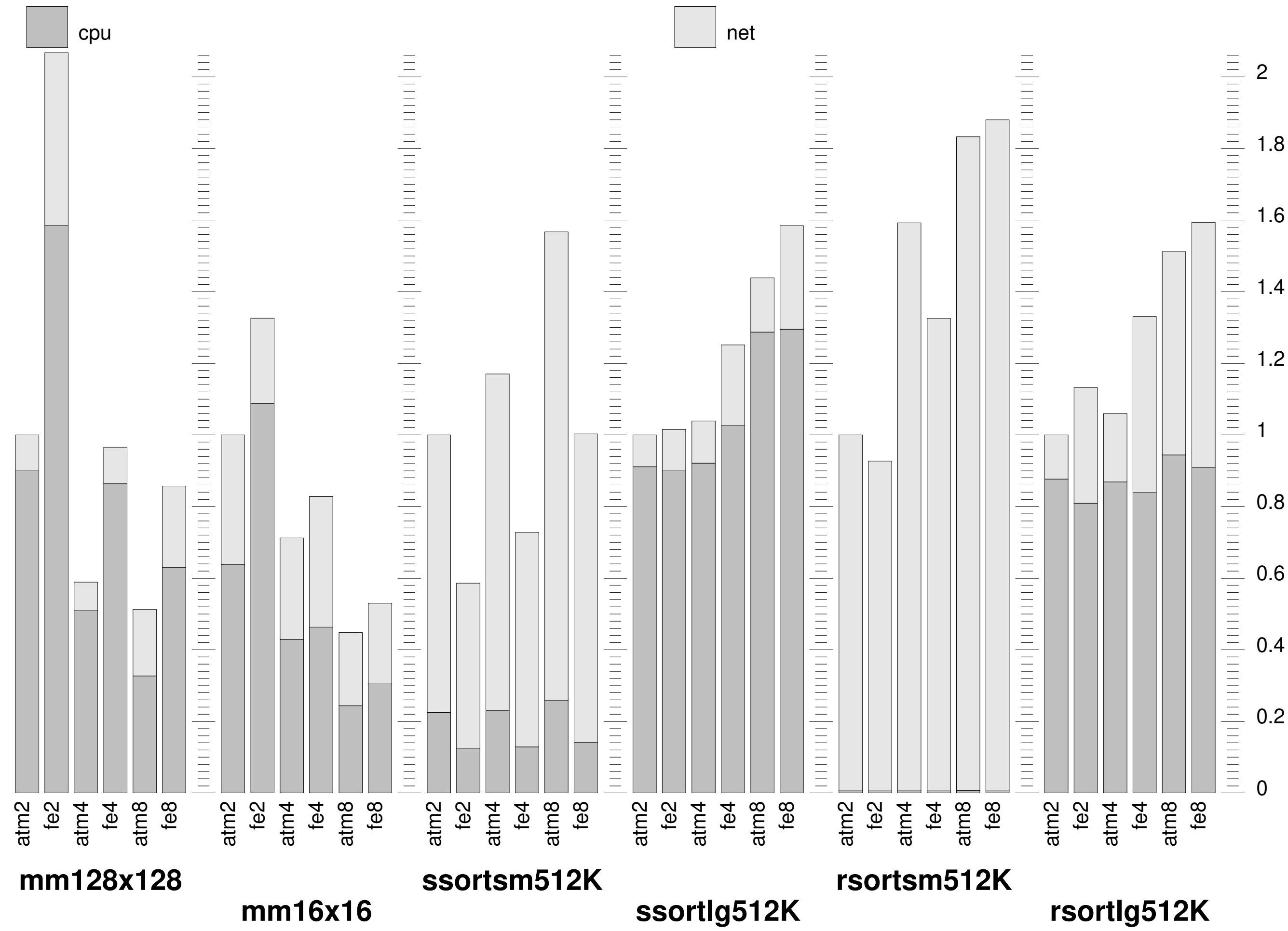


Figure 7: Relative execution times of 6 Split-C benchmarks. The execution times on a 2-node ATM cluster is used as the reference and times for Fast Ethernet as well as ATM clusters of 2, 4, and 8 nodes are shown. The Fast Ethernet cluster consists of Pentium PCs running Linux while the ATM cluster uses Sparcstations with 140Mbps ATM. The execution times are divided into the time spent in computation (cpu) and communication (net) intensive parts.

receive overhead is dominated by the message copy into the appropriate user buffer. The i960 co-processor on the ATM interface is significantly slower than the Pentium host and its use slows down the latency times. The main benefit of the co-processor is to allow the network interface to examine the packet header and DMA the data directly into the correct user-space buffer, thereby eliminating a costly copy.

Split-C application benchmarks have been used to demonstrate that inexpensive Pentium workstation clusters can be employed for parallel computing with U-Net/FE as the basic interconnect. While applications requiring higher bandwidth may fare better with an ATM interconnect, Fast Ethernet provides an important price/performance point for workstation clusters.

7 Acknowledgments

The authors would like to thank Donald Becker of the Beowulf Project at CESDIS for sharing his Linux kernel driver for the DC21140 and Padma Venkataramanan at FORE Systems for helping us to understand the i960 byte-swapping issues with the FORE PCA-200PC ATM interface. Grzegorz Czajkowski and Chi-Chao Chang at Cornell provided help with the Split-C benchmark suite.

The U-Net project is supported by the Air Force Material Contract F30602-94-C-0224 and ONR contract N00014-92-J-1866. Thorsten von Eicken is supported by a fellowship from the Sloan Foundation. The Pentium systems used in the cluster were donated by the Intel Corporation.

8 References

[1] T.E. Anderson, D.E. Culler, D.A. Patterson, et. al. *A Case for NOW (Networks of Workstations).* IEEE Micro, Feb. 1995, pages 54-64.

[2] D. Becker, T. Sterling, D. Savarese, B. Fryxell, and K. Olson. *Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation.* In Proc. of the 4th HPDC ‘95.

[3] M. Blumrich, C. Dubnicki, E. W. Felten and K. Li. *Virtual-Memory-Mapped Network Interfaces*. IEEE Micro, Feb. 1995, pages 21-28.

[4] D. Boggs, J. Mogul, and C. Kent. *Measured Capacity of an Ethernet: Myths and Reality.* WRL Research Report 88/4, Western Research Laboratory, September 1988.

[5] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. *Introduction to Split-C.* In Proc. of Supercomputing '93.

[6] D. E. Culler, A. Dusseau, R. Martin, K. E. Schauser. *Fast Parallel Sorting: from LogP to Split-C.* In Proc. of WPPP '93, July 93.

[7] D.E. Culler, et. al. *Generic Active Message Interface Specification, version 1.1.* http://now.cs.berkeley.edu/Papers/Papers/gam_spec.ps

[8] P. Druschel and L. Peterson. *Fbufs: A High-Bandwidth Cross-Domain Transfer Facility.* In Proc. of the 14th SOSP. pages 189-202. December 1993.

[9] P. Druschel, L. Peterson, and B.S. Davie. *Experiences with a High-Speed Network adapter: A Software Perspective.* In Proc. of SIGCOMM-94, pages 2-13, Aug 1994.

[10] A. Edwards, G. Watson, J. Lumley, D. Banks, C. Calamvokis and C.Dalton. *User-space protocols deliver high performan*ce to applications on a low-cost Gb/s LAN. In Proc. of SIGCOMM-94, pages 14-23, Aug. 1994

[11] J. Kay and J. Pasquale. The importance of Non-Data Touching Processing Overheads. In Proc. of SIGCOMM-93, pages 259-269, Aug. 1993

[12] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In Proc. of Supercomputing '95, San Diego, California.

[13] R. Seifert. The Effect of Ethernet Behavior on Networks using High-Performance Workstations and Servers. http://wwwhost.ots.utexas.edu/ethernet/pdf/techrept13.pdf

[14] C. A. Thekkath, H. M. Levy, and E. D. Lazowska. Separating Data and Control Transfer in Distributed Operating Systems. In Proc. of the 6th Int’l Conf. on ASPLOS, Oct 1994.

[15] T. von Eicken, A.Basu and V.Buch. Low-Latency Communication Over ATM Networks Using Active Messages. IEEE Micro, Feb. 1995, pages 46-53.

[16] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In Proc. of the 15th ACM SOSP, pages 40-53, December 1995.

[17] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In Proc. of the 19th ISCA, pages 256-266, May 1992.

[18] T. M. Warschko, W. F. Tichy, and C. H. Herter. Efficient Parallel Computing on Workstation Clusters. http://wwwipd.ira.uka.de/~warschko/parapc/sc95.html

[19] M. Welsh, A. Basu, and T. von Eicken. Low-latency communication over Fast Ethernet. In Proc. of EuroPar ‘96, Lyon, France, August 1996.

[20] J. Wilkes. An interface for sender-based communication. Tech. Rep. HPL-OSR-92-13, Hewlett-Packard Research Laboratory, Nov. 1992.