Low-Latency Communication over Fast Ethernet

Matt Welsh, Anindya Basu, and Thorsten von Eicken
{mdw,basu,tve}@cs.cornell.edu

Department of Computer Science
Cornell University, Ithaca, NY 14853

http://www.cs.cornell.edu/Info/Projects/U-Net

Abstract

Fast Ethernet (100Base-TX) can provide a low-cost alternative to more esoteric network technologies for high-performance cluster computing. We use a network architecture based on the U-Net approach to implement low-latency and high-bandwidth communication over Fast Ethernet, with performance rivaling (and in some cases exceeding) that of 155 Mbps ATM. U-Net provides protected, user-level access to the network interface and enables application-level round-trip latencies of less than 60µs over Fast Ethernet.

1 Introduction

High-performance computing on clusters of workstations requires low-latency communication in order to efficiently implement parallel languages and distributed algorithms. Recent research [1, 6, 8] has demonstrated that direct application access to the network interface can provide both low-latency and high-bandwidth communication in such settings and is capable of showing performance comparable to state-of-the-art multiprocessors. Previous work in this area has concentrated on high-speed networks such as ATM, the technology for which is still emerging and somewhat costly. This paper presents U-Net/FE, a user-level network architecture employing low-cost Fast Ethernet (100Base-TX) technology.

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 assured through the virtual memory system and kernel control of connection set-up and tear-down. A previous implementation of U-Net over ATM[6] demonstrated that this architecture is able to efficiently support low-latency communication protocols such as Active Messages[7] and parallel languages such as Split-C[2]. However, 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 adaptor (as in the ATM implementation) is a necessary part of the design.

The U-Net/FE implementation described here demonstrates directly that U-Net can indeed be implemented efficiently over a network substrate other than ATM. The performance results show that low-latency communication over 100Mbps Fast Ethernet is possible using off-the-shelf hardware components. As a result, the cost of workstation clusters is brought down through the use of inexpensive personal computers and a commodity interconnection network.

1.1 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[5] 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[4] implementation achieves high performance on a Myrinet network using communication primitives similar to Active Messages. The network interface is accessed directly from user-space but without providing support for simultaneous use by multiple applications. The HP Hamlyn[9] 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[1] 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[8] 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; however, this does not include the round-trip latency.

2 U-Net user-level communication architecture

The U-Net architecture[6] 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. The U-Net platform is in itself is not dependent on the underlying hardware. 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.

2.1 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 as well as *message queues* to hold descriptors for 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.

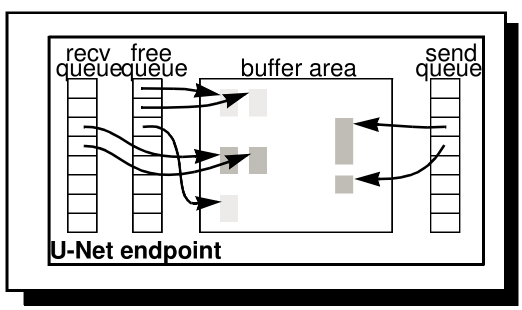


Figure 1: U-Net building blocks.

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 picks-up the message and inserts it into the network. The management of the 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.

Incoming messages are demultiplexed by U-Net based on a tag in each incoming message to determine its destination endpoint and thus the appropriate buffer area for the data and message queue for the descriptor. The exact form of this message tag depends on the network substrate; for example, in an ATM network the virtual channel identifiers (VCIs) may be used. A process registers these tags with U-Net by creating communication channels: on outgoing messages the channel identifier is used to place the correct tag into the message (as well as possibly the destination address or route) and on incoming messages the tag is mapped into a channel identifier to signal the origin of the message to the application. An operating system service needs 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.

After demultiplexing, the data is transferred into one or several free buffers and a message descriptor with pointers to the buffers is pushed onto the receive queue. As an optimization for small messages—which are used heavily as control messages in protocol implementations—the receive queue may hold entire small messages in descriptors. Note that the application cannot control the order in which receive buffers are filled with incoming data.

3 Fast Ethernet Implementation of U-Net

U-Net/FE was implemented on a 133Mhz Pentium system running Linux and using the DECchip DC21140 network interface. The DC21140 is a PCI busmastering Fast Ethernet controller capable of transferring complete frames to and from host memory via DMA. The controller includes a few on-chip control and status registers, a DMA engine, and a 32-bit Ethernet CRC generator/checker. It maintains circular send and receive rings containing descriptors which point to buffers for data transmission and reception in host memory. This interface was designed for traditional in-kernel networking layers in which the network interface is controlled by a single agent on the host. As a result the DC21140 lacks any mechanisms which would allow direct user-level access to the chip without compromising protection. This means that U-Net has to be implemented in the kernel: a device driver and a special trap are used to safely multiplex the network interface among multiple applications.

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, free, 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.

3.1 Endpoint and Channel Creation

Creation of user endpoints and communication channels is managed by the operating system. This is necessary to enforce protection boundaries between processes and to properly manage system resources. Endpoint creation consists of issuing an *ioctl* to

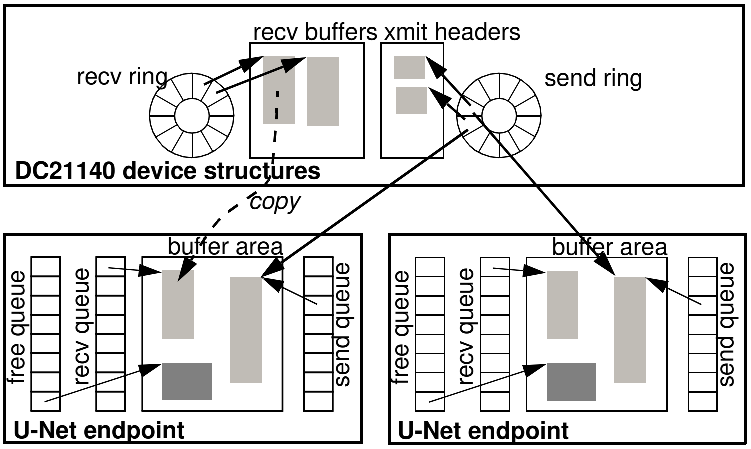


Figure 2: U-Net/FE endpoint and device data structures.

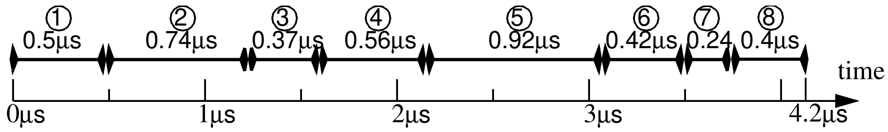
the U-Net device driver requesting space for the message queues and buffer areas. The kernel allocates a segment of pinned-down physical memory for the endpoint, which is mapped into the process address space by use of an *mmap* system call.

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. A communication channel can be created by issuing an *ioctl* 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; the port ID is used to demultiplex incoming messages to a particular endpoint. The operating system registers the requested addresses and returns a channel tag to the application. The channel tag 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 tag when placing new entries on the receive queue for the application.

3.2 Transmit

When the user process wishes to transmit data on the network, it first constructs the message in the buffer area and then pushes an entry onto the U-Net send queue, specifying the location, size, and transmit channel tag for each buffer to send. The DC21140 device send ring is shared between all endpoints and must therefore be managed by the kernel. The user process 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 133MHz 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 one being an in-kernel buffer with the Ethernet header and packet length field, and the second being the user buffer containing the



1. trap entry overhead
2. U-Net send param check
3. Ethernet header set-up
4. device send ring descr set-up
5. issue poll demand to DC21140
6. free send ring descr of prev message
7. free U-Net send queue entry of prev message
8. return from trap

Figure 3: Fast Ethernet transmission time-line for a 40 byte message (66 bytes with the Ethernet and U-Net header).

data (for multi-buffer user messages additional descriptors are used). By pointing directly to the U-Net buffer a copy is avoided and the DC21140 transfers the data directly from user-space into the network.

The in-kernel buffers for the ethernet header are preallocated and protected from user-access. Each buffer holds 16 bytes: 14 are filled by the service routine with the Ethernet header for the appropriate U-Net channel, and two contain the actual length of the user message. This last field is needed at the receiving end to identify the exact length of messages under the minimum Ethernet payload size of 46 bytes.

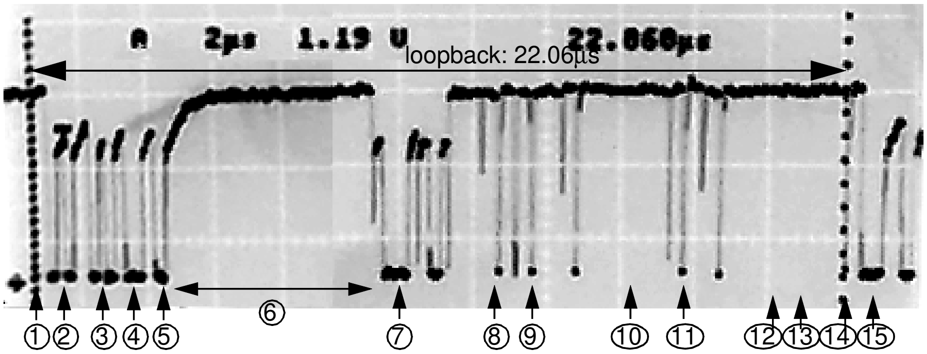
After all the descriptors have been pushed onto the device transmit ring the service routine issues a transmit poll demand to the DC21140. The latter processes the transmit ring, adds padding for small payloads if necessary, and adds the 32-bit packet CRC. Once the frame has been serialized onto the wire, the DC21140 sets a bit in the transmit ring descriptor to signal that the transmission is complete. The kernel service routine uses this information to mark the associated U-Net send queue entries as free.

Figure 3 shows the time-line of a kernel send trap for a 40-byte message which, with the Ethernet and U-Net headers, corresponds to a 66-byte frame. The DC21140 engages the DMA to access the device send ring descriptor after it receives the host's poll demand (i.e. towards the end of segment 5 in the figure). The transfers on the PCI bus are shown in Figure 4 which depicts an oscilloscope screen shot taken of the active-low PCI *FRAME* signal during a loop-back message transmission. The five accesses at the left show that the DC21140 rapidly fetches the message data from the two buffers and serializes it onto the wire. The transmission of the Ethernet frame takes 5.4µs after which the first data of the loop-back message is DMA-ed back into a receive buffer. A timing analysis of the U-Net trap code shows that the processor overhead required to push a message into the network is approximately 4.1µs of which about 20% are consumed by the trap overhead.

3.3 Receive

Upon packet receipt the DC21140 transfers the data into buffers in host memory pointed to by a device receive ring analogous to the transmit ring. The controller then checks the CRC of the incoming packet and interrupts the host, which consumes new entries on the device receive ring and hands them back to the DC21140.

The kernel interrupt routine determines the destination endpoint and channel tag from the U-Net port number contained in the Ethernet header, copies the data into the

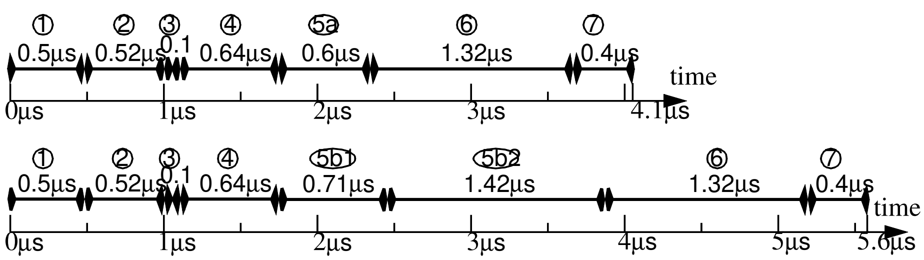


1. poll demand to DC21140
2. DMA fetch of tx descr
3. DMA fetch of header buffer
4. DMA fetch of data buffer
5. DMA prefetch of next tx descr
6. message transmission and reception
7. DMA of incoming message
8. DMA of tx descr (completion)
9. intr start - check DC21140
10. copy message
11. return from interrupt check DC21140
12. user-level recv
13. xmit of next message
14. next poll demand
15. DMA fetch of tx descr

Figure 4: Oscilloscope screen shot of a loop-back transmission and reception of a 66-byte Ethernet frame on the DC21140. The trace shows the PCI bus cycle FRAME signal (active low).

appropriate U-Net buffer area and enqueues an entry in the user receive queue. As an optimization, small messages (under 56 bytes) are copied directly into the U-Net receive descriptor itself.

Figure 5 shows the time line for 40 and 100-byte messages. The short message optimization is effective in that it saves over 15% by skipping the allocation of a receive buffer. For messages of more than 64 bytes the copy time increases by 1.42µs for each additional 100 bytes. The latency between frame data arriving in memory and invocation of the interrupt handler is roughly 2µs and the major cost of the receive interrupt handler is the additional memory copy required to place incoming data into the appropriate user buffer area.



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

Figure 5: Fast Ethernet reception time-line for a 40-byte and a 100-byte message.

3.4 Latency and Bandwidth Performance

Figure 6 depicts the application-to-application message round-trip time as a function of message size for U-Net/FE on the DC21140 and compares it to the ATM implementation of U-Net on the FORE Systems PCA-200 ATM interface[1]. Message sizes range from 0 to 1498 bytes for U-Net/FE and ATM. 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 16-port switch. The round-trip time for a 40-byte message over Fast Ethernet rages 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.

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 for ATM.

Figure 7 shows the bandwidth over the raw U-Net interface for Fast Ethernet and ATM in Mbits/sec for message sizes ranging from 0 to 1498 bytes. For messages as small as 1Kbyte the bandwidth approaches the peak of about 97Mbps (taking into 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 135 Mbps.

4 Summary

The U-Net/FE architecture has been presented as an efficient user-level communication mechanism for use over 100Mbit Fast Ethernet. This system provides low-latency

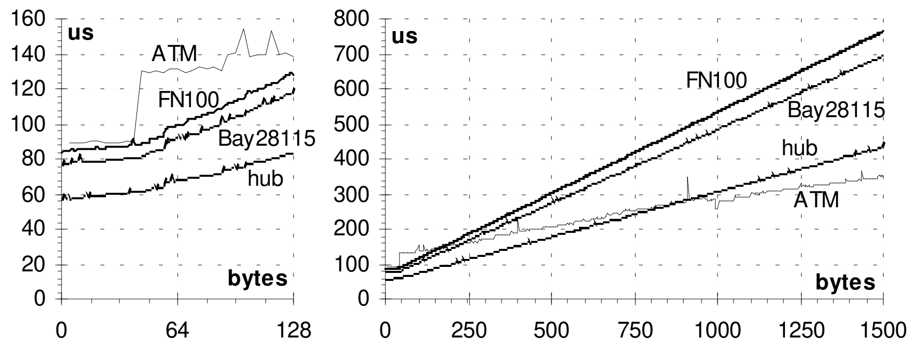


Figure 6: Round-trip message latency vs. message size for Fast Ethernet and ATM.

1. The Fore Systems PCA-200 ATM network interface includes an on-board i960 processor to perform the segmentation and reassembly of cells as well as DMA to/from host memory. The i960 processor is controlled by firmware (downloaded from the host) which implements U-Net directly: for message transmission user applications push descriptors directly into i960 memory and incoming messages are DMA-ed straight into user-space.
2. A previous implementation of U-Net over ATM on 140Mbps TAXI demonstrated 65msec round-trip latency; however, here the physical medium is 155Mbps OC-3 and considerable additional overhead is incurred due to SONET framing.

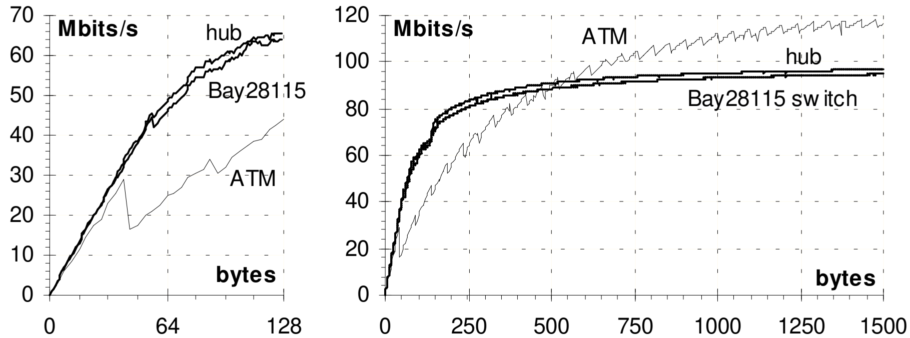


Figure 7: Bandwidth vs. message size for Fast Ethernet and ATM.

communication with performance rivaling that of ATM. For small messages the round-trip latency is in fact less than that for 155Mbps ATM which demonstrates that Fast Ethernet can be employed as a low-latency interconnect for workstation clusters.

This paper has also shown that the U-Net architecture as implemented on ATM can be extended to other networks and network interfaces. In particular, a kernel trap can be used successfully in case the network interface hardware is not capable of multiplexing/demultiplexing messages directly.

5 Acknowledgments

The authors thank Donald Becker of the Beowulf Project at CESDIS for sharing his Linux kernel driver for the DC21140. The U-Net project is supported by the Air Force Material Contract F30602-94-C-0224 and ONR contract N00014-92-J-1866.

6 References

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

[2] 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.

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

[4] 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.

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

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

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

[8] 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

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