U-Net: A User-Level Network Interface for Parallel and Distributed Computing

Computer Science Technical Report "to appear"
DRAFT — Comments welcome

Anindya Basu, Vineet Buch, Werner Vogels, Thorsten von Eicken

Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract

The U-Net communication architecture provides processes with a virtual view of a network device to enable user-level access to high-speed communication devices. The architecture, implemented on standard workstations using off-the-shelf ATM communication hardware, removes the kernel from the communication path, while still providing full protection.

The model presented by U-Net allows for the construction of protocols at user level whose performance is only limited by the capabilities of network. The architecture is extremely flexible in the sense that traditional protocols like TCP and UDP, as well as novel abstractions like Active Message can be implemented efficiently. A U-Net prototype on an 8-node ATM cluster of standard workstations achieves 15Mbytes/s TCP bandwidth with 1Kbyte buffers and demonstrates performance equivalent to Meiko CS-2 and TMC CM-5 supercomputers on a set of Split-C benchmarks.

1 Introduction

The increased availability of high-speed local area networks has shifted the bottleneck in local-area communication from the limited bandwidth of network fabrics to the software path traversed by messages at the sending and receiving ends. In particular, in a traditional UNIX networking architecture, the path taken by messages through the kernel involves several copies and crosses multiple levels of abstraction between the device driver and the user application. The resulting processing overheads limit the peak communication bandwidth and cause high end-to-end message latencies. The effect is that users who upgrade from ethernet to a faster network fail to observe an application speed-up commensurate with the improvement in raw network performance. A solution to this situation seems to elude vendors to a large degree because many fail to recognize the importance of per-message overhead and concentrate on peak bandwidths of long data streams instead. While this may be justifiable for a few applications such as video playback, most applications use relatively small messages and rely heavily on quick round-trip requests and replies. The increased use of techniques such as distributed shared memory, remote procedure calls, remote object-oriented method invocations, and distributed cooperative file caches will further increase the importance of low round-trip latencies and of high bandwidth at the low-latency point.

The use of clusters of workstations interconnected by a high-speed LAN for new application domains also increases the demand for new network protocols. The traditional networking architecture which places all protocol processing into the kernel cannot provide the flexibility required for such demands. For example, the transmission of MPEG compressed video streams can greatly benefit from customized retransmission protocols which embody knowledge of the real-time demands as well as the interdependencies among video frames[REF].

One of the most promising techniques to improve networking layer performance on workstation-class machines is to move parts of the protocol processing into user space. This paper argues that in fact the entire

For further information, email tve@cs.cornell.edu or browse http://www.cs.cornell.edu/Info/Projects/U-Net/
The software described in this paper will be made available in source form, except for the SBA-200 firmware which can only be distributed as object code.

protocol stack should be performed at user level and that the operating system and hardware should allow protected user-level access directly to the network. The goal is to remove the kernel completely from the critical path and to allow the communication layers used by each process to be tailored to its demands. The key issues that arise are

- multiplexing the network among processes,
- providing protection such that processes using the network cannot interfere with each other,
- managing limited communication resources without the aid of a kernel path, and
- designing an efficient yet versatile programming interface to the network.

Some of these issues have been solved in more recent parallel machines such as in the CM-5, the Meiko CS-2, and the IBM SP-2, all of which allow user-level access to the network. However, all these machines have a custom network and network interface, and they usually restrict the degree or form of multiprogramming permitted on each node. This implies that the techniques developed in these designs cannot be applied to workstation clusters directly.

This paper describes the U-Net architecture for user-level communication on an off-the-shelf hardware platform (SparcStations with Fore Systems ATM interfaces) running a standard operating system (SunOS 4.1.3). The communication architecture virtualizes the network device so that each process has the illusion of owning the interface to the network. Protection is assured through kernel control of connection setup and tear-down. The U-Net architecture is able to support both legacy protocols and novel networking abstractions: TCP and UDP as well as Active Message are implemented and exhibit performance that is only limited by the processing capabilities of the network interface. Using Split-C, a state-of-the-art parallel language, the performance of seven benchmark programs on an ATM cluster of standard workstations rivals that of current parallel machines. In all cases U-Net was able to expose the full potential of the ATM network by saturating the 140Mbits/sec fiber, using either traditional networking protocols or advanced parallel computing techniques.

The major contributions of this paper are to propose a simple user-level communication architecture (Sections 2 and 3) which is independent of the network interface hardware (i.e. allows many hardware implementations), to describe two high-performance implementations on standard workstations (Sections 4 and 5), and to evaluate its performance characteristics for TCP and RPC communication (Section 6) as well as for communication in parallel programs (Sections 7 and 8).

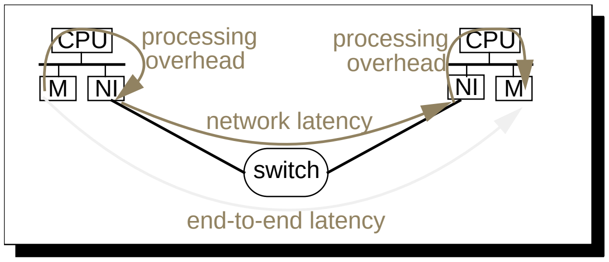

Figure 1: Processing overhead, network latency, and end-to-end latency.

While other researchers have proposed user-level network interfaces independently, this is the first presentation of a full system which does not require custom hardware or OS modification and which supports traditional networking protocols as well as state of the art parallel language implementations. Since it exclusively uses off-the-shelf components, the system presented here establishes a baseline to which more radical proposals that include custom hardware or new OS architectures must be compared to.

2 U-Net Motivation and Design

The U-Net architecture focuses on reducing the processing overhead required to send and receive messages. In addition, it provides flexible access to the lowest layers of the network. The intent is to enable the use of clusters of workstations for applications that require low-latency communication, to reduce the cost of achieving high bandwidth, and to facilitate the use of novel communication protocols.

The term *processing overhead* is used here to refer to the time spent by the processor in handling messages at the sending and receiving ends. This may include buffer management, message copies, checksumming, flow-control handling, interrupt overhead, as well as controlling the network interface. As shown in Figure 1, separating this overhead from the *network latency* distinguishes the costs stemming from the network fabric technology from those due to the networking software layers.

Recent advances in network fabric technology have dramatically improved network bandwidth while the network latency and the processing overheads have not been affected nearly as much. The effect is that for large messages, the *end-to-end latency*—the time from the source application executing “send” to the time the destination application receiving the message—is dominated by the transmission time and thus the new networks offer a net improvement. For small messages, however, the processing overheads dominate and the improvement in transmission time is not only insignificant in comparison but also offset by the introduction of switches into the network.

2.1 The Importance of low-overhead low-latency communication

Against the backdrop of ever improving performance for large messages, small messages are becoming increasingly important in many applications. For example, in distributed systems:

- Object-oriented technology is finding wide-spread adoption and is naturally extended across the network by allowing the transfer of objects and the remote execution of methods (e.g., CORBA and the many C++ extensions). Objects are generally small, relative to the message sizes required for high bandwidth (around 100 bytes vs. several Kbytes) and thus communication performance suffers unless message overhead is low.
- The electronic workplace relies heavily on sets of complex distributed services which are intended to be transparent to the user. The majority of such service invocations are requests to simple database servers that implement mechanisms like object naming, object location, authentication, protection, etc. The message size seen in these systems range from 20-80 bytes for the requests and the responses generally can be found in the range of 40-200 bytes.
- To limit the network traversal of larger distributed objects, caching techniques have become a fundamental part of most modern distributed systems. Keeping the copies consistent introduces a large number of small coherence messages. The round-trip times are important as the requestor is usually blocked until the synchronization is achieved.
- Software fault-tolerance algorithms and group communication tools often require multi-round protocols, the performance of which is latency-limited. Low-latency communication prevents such protocols to be used today in process-control applications, financial trading systems, or multimedia groupware applications.

Without projecting into the future, existing more general systems can benefit substantially as well:

- Reliable data stream protocols like TCP have buffer requirements that are directly proportional to the round-trip end-to-end latency. For example the TCP window size is the product of the network bandwidth and the round-trip time. Achieving low-latency will keep the buffer consumption within reason and thus make it feasible to achieve maximal bandwidth.
- Numerous client/server architectures are based on a RPC style of interaction, by drastically improving the communication latency for requests, responses and their acknowledgments, a large number of systems may see significant performance improvements.
- Although remote file systems are often categorized as bulk transfer systems, they depend heavily on the performance of small messages. A week-long trace of all NFS traffic to the departmental CS fileserver at UC Berkeley has shown that the vast majority of the messages is under 200 bytes in size and that these messages account for roughly half the bits sent[1].

In addition, many researchers propose to use networks of workstations to provide the resources for compute intensive parallel applications. In order for this to become feasible, the communication costs across LANs must reduce by more than an order of magnitude to be comparable to those on modern parallel machines.

2.2 The bottleneck in traditional networking architectures

The introduction of ATM networks to standard off-the-shelf workstations promised high bandwidth links as well as low network latency due to the small ATM cells, avoiding the high-latency problems that arose with FDDI. Experiments, however, show that ATM networks plugged into the traditional networking architectures fail to meet this promise. Although high bandwidth can be achieved, this is only possible under unreliable conditions and only when using large buffers and large messages.

In fact, the end-to-end latency for small messages is worse over ATM than over Ethernet and does not at all mirror the capabilities of the underlying network. A preliminary analysis of the interaction between the Fore SBA-200 ATM interface and the SunOS 4.1.3 kernel reveals that the main additional processing overhead is in buffer handling at the device control level. At both the sending and the receiving end a number of costly operations have to be performed to match the device buffer and memory abstractions with the kernel buffers (mbufs) and to handle data alignment restrictions. The failure of the operating system software to exploit the capabilities of the ATM network can be attributed to a large degree to the use of generalized buffer and data transfer strategies.

In summary, a new abstraction for high-performance communication is required to deliver the promise of low-latency, high-bandwidth communication to the applications on standard workstations using off-the-shelf networks.

2.3 Towards a new networking architecture

The central idea in the new abstraction is to simply remove the kernel from the critical path of sending and receiving messages. This eliminates the system call overhead, and more importantly, offers the opportunity to streamline the buffer management which can now be performed at user-level. As several research projects have pointed out, eliminating the kernel from the send and receive paths requires that some form of a message multiplexing and demultiplexing device (in hardware or in software) is introduced for the purpose of enforcing protection boundaries.

The approach proposed in this paper is to incorporate this mux/demux directly into the network interface (NI), as depicted in Figure 2, and to move all buffer management and protocol processing to user-level. This, in essence, virtualizes the NI and provides each process the illusion of owning the interface to the network. Such an approach raises the issues of selecting a good virtual NI abstraction to present to processes, of providing support for legacy protocols side-by-side with next generation parallel languages, and of enforcing protection without kernel intervention on every message.

2.4 Related work

A number of the issues surrounding user-level network interface access have been studied in the past. For the Mach3 operating system a combination of a powerful message demultiplexer in the microkernel, and a user-level implementation of the TCP/IP protocol suite solved the network performance problems that arose when the Unix OS-Server was responsible for all network communication. The performance achieved is roughly the same as that of a monolithic BSD system.[13]

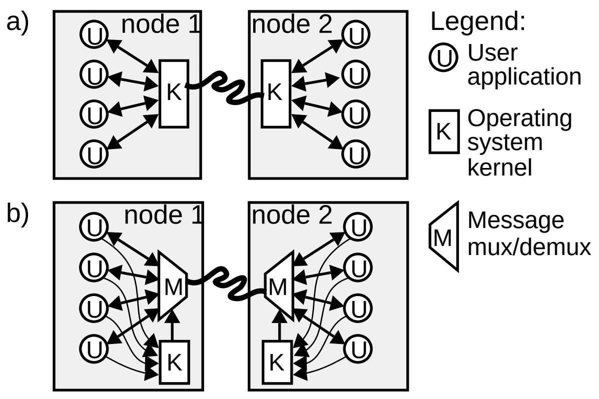

Figure 2: The traditional networking architecture (a) places the kernel in the path of all communication. The U-Net architecture (b) only uses a simple multiplexing/demultiplexing agent—that can be implemented in hardware—in the data communication path and uses the kernel only for set-up.

More recently, the *application device channel* abstraction, developed at the University of Arizona, provides application programs with direct access to the experimental OSIRIS ATM board[8]. A packet filter style demultiplexer, implemented in hardware, combined with the *fbufs* cross-domain buffer management[7] is developed to achieve the high bandwidth data transfers that are the goal of the project. [Need more details]

The software architecture built to support the HP experimental *Jetstream* LAN[9] (800 Mbits, timed-token LAN) also makes provisions (with support of the *Afterburner* board) for user-level communication protocols. The main focus of the project is on support for high bandwidth (>200 Mbit) byte stream applications. [Need more details]

In the parallel computing community recent machines (e.g., Thinking Machines CM-5, Meiko CS-2, IBM SP-2, Cray T3D) provide user-level access to the network, but the solutions rely on custom hardware and are somewhat constrained to the controlled environment of a multiprocessor. On the other hand, given that these parallel machines resemble clusters of workstations ever more closely, it is reasonable to expect that some of the concepts developed in these designs can indeed be transferred to workstations.

Successive simplifications and generalizations of shared memory is leading to a slightly different type of solution in which the network can be accessed indirectly through memory accesses. Shrimp[2] uses custom NIs to allow processes to establish channels connecting virtual memory pages on two nodes such that data written into a page on one side gets propagated automatically to the other side. Thekkath[16] proposes a memory-based network access model that separates the flow of control from the data flow. The remote memory operations have been implemented by emulating unused opcodes in the MIPS instruction set. While the use of a shared memory abstraction allows a reduction of the communication overheads, it is not clear how to efficiently support legacy protocols, long data streams, or remote procedure call.

2.5 U-Net design goals

The first and predominant goal of the U-Net architecture is to achieve high-performance low-latency communication. More specifically, the following performance criteria should be met:

- the latency experienced by the application is dominated by the time the messages spend on the wire,
- low-latency is achieved by minimizing the send and receive overheads, and
- high bandwidth is achieved for small messages with-

out sacrificing latency.

What sets U-Net most apart from the proposals discussed above are the concept that the above goals should be achieved on widely available standard workstations using off-the-shelf communication hardware, and the fact that U-Net builds a foundation that supports legacy as well as innovative protocols. This means that the architecture should be designed to:

- satisfy traditional protocols and simultaneously open doors for ubiquitous use of new communication abstractions such as *Active Messages[18]*,
- be simple, understandable and controllable, but general enough to be applicable to other classes of computer architectures and network types such as Myrinet, and
- remain independent of any particular protocol, protocol abstraction or implementation method.

3 The user-level network interface architecture

The user-level network interface (U-Net) architecture is inspired by the facilities provided by many DMA-capable ethernet and FDDI controllers in use today. It simplifies and virtualizes the interface in such a way that a combination of operating system and hardware mechanisms can provide every process[1] 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 memory locations 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 as well as resource consumption limits. In particular, a process has control over both the contents of each message (with the exception of the source and destination addresses used for protection) and the management of send and receive resources, such as buffers.

3.1 Building blocks

The U-Net architecture is composed of three main building blocks, shown in Figure 3: *communication segments* which are regions of memory that hold message data, *message queues* which hold descriptors for messages that have been received or that are to be sent, and *endpoints* which serve as the basic addressing unit across the network. Each process that wishes to access the network first creates one or more endpoints, then

1. The terms “process” and “application” are used interchangeably to refer to arbitrary unprivileged UNIX processes.

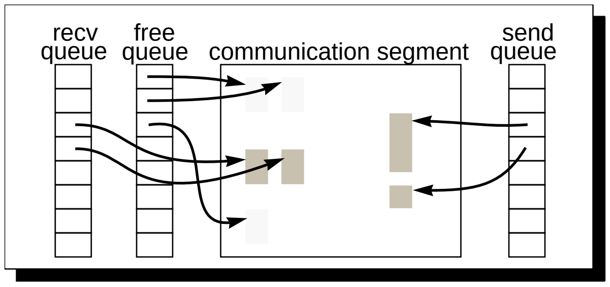

Figure 3: U-Net architecture building blocks.

associates a communication segment and a set of *send*, *receive*, and *free* message queues with each endpoint. Multiple communication segments can be used to keep the buffer management of different communication channels disjoint, and multiple endpoints may share a single set of queues[2]. After this set-up, the name of each endpoint serves as a network-wide address to which messages can be sent and from which messages can originate. The mechanism used by processes to find out about each other’s endpoint addresses is external to the U-Net architecture.

Endpoints also serve as the unit of protection among multiple processes accessing the network as well as across the network. This is achieved using three mechanisms:

- endpoints, communication segments, and message queues are only accessible by the owning process,
- outgoing messages are tagged with the originating endpoint address and incoming messages are demultiplexed by U-Net and delivered to the correct destination endpoint, and
- a per-endpoint access control list (ACL) restricts from which endpoints incoming messages are accepted, thus preventing unauthorized senders to consume all receive resources.

3.2 Sending messages

To send a message, a user process composes the data in the communication segment and pushes a descriptor for the message onto the send queue. At that point, the network interface is expected to pick the message up and insert it into the network. If the network is backed-up, the network interface will simply leave the descriptor in the queue and eventually exert back-pressure to the user process when the queue becomes full.The NI provides a mechanism to indicate whether a message in the queue has been injected into the network, typically by setting a flag in the descriptor and providing random read access to the entire queue.

2. This is encouraged because multiple sets of queues tend to increase the overhead of servicing the network.

To avoid deadlock, the back-pressure mechanism requires a careful contract between processes and U-Net: the network must guarantee deadlock free-ness under the condition that all communicating parties eventually pop all message descriptors from their receive queues, in particular no process may indefinitely attempt to push a message onto a full send queue without accepting incoming messages.

3.3 Receiving messages

Incoming messages are demultiplexed by U-Net based on the destination endpoint address: the data is transferred into the appropriate communication segment, and a message descriptor is pushed onto the corresponding receive queue. The receive model supported by U-Net is either polling or event driven: the process can periodically check the status of the receive queue or it can register an upcall[1] with U-Net. These upcalls are used by the U-Net layer to signal that the state of the receive queue satisfies a specific condition. The three conditions supported by U-Net are: the receive queue is non-empty, the receive queue is almost full, and the receive queue has had a message pending for a while. The first one allows event driven reception. The second allows processes to be awakened before the receive queue overflows. The third one allows the application to poll at irregular intervals and to be notified if a message arrives during a period of non-polling[2]. U-Net does not specify the nature of the upcalls which could be UNIX signal handlers, threads, or user-level interrupt handlers.

In order to amortize the cost of an upcall over the reception of several messages it is important that a U-Net implementation allow all messages pending in the receive queue to be consumed in a single upcall. Furthermore, a process must be able to disable upcalls cheaply in order to form critical sections of code that are atomic relative to message reception.

The upcalls are also used to signal error conditions to the application. On the sending side an error upcall indicates a serious problem such as an illegal message descriptor contents or a fatal network error (e.g., network/node unreachable). On the receiving end upcalls are used to signal the arrival of corrupted messages, the reception of messages from endpoints not permitted by the ACL associated with the destination endpoint, and the overflow of receive resources.

1. The term “upcall” is used in a very general sense to refer to a mechanism which allows U-Net to signal an asynchronous event to the application.
2. This mode is useful in parallel programs where the compiler generates polls automatically but certain functions are linked from sequential libraries and therefore do not include polls.

3.4 Zero-copy vs. true zero-copy

As in all high performance networking architectures one of the main challenges is minimizing copying of message data. U-Net attempts to support a “true zero copy” architecture in which data can be sent directly out of the application data structures without intermediate buffering and where the NI can transfer arriving data directly into user-level data structures as well. In consideration of current limitations on I/O bus addressing and on NI functionality, the U-Net architecture specifies two levels of sophistication: a *base-level* which requires an intermediate copy into a networking buffer and corresponds to what is generally referred-to as zero copy, and a *direct-access* U-Net which supports true zero copy without any intermediate buffering.

The base-level U-Net architecture matches the operation of existing network adapters closely by providing a reception model based on a queue of free buffers that are filled by U-Net as messages arrive. It also regards communication segments as a limited resource and places an upper bound on their size such that it is not feasible to regard communication segments as memory regions in which general data structures can be placed. This means that for sending each message must be constructed in a buffer in the communication segment and on reception data is deposited in a similar buffer. This corresponds to what is generally called “zero-copy”, but which in truth represents one copy, namely between the application’s data structures and a buffer in the communication segment.

Direct-access U-Net supports true zero copy protocols by allowing communication segments to span the entire process address space and by letting the sender specify an offset within the destination communication segment at which the message data is to be deposited directly by the NI. The difficulties in implementing direct-access come from the fact that it requires (i) the NI to include some form of memory mapping hardware, (ii) all of physical memory to be addressable from the NI, and (iii) page faults on message arrival to be handled appropriately.

The U-Net implementations described here support the base-level architecture because the hardware available does not support the memory mapping required for the direct-access architecture. In addition, the bandwidth of the ATM network used does not warrant the enhancement because the copy overhead is not a dominant cost. The following subsections describe the base-level and direct-access U-Net architectures, as well as two base-level implementations on an ATM cluster of workstations.

3.5 Base-level U-Net architecture

The base-level U-Net architecture supports a queue-based interface to the network which stages messages in a limited-size communication segment on their way between the network and application data structures. The communication segments are allocated by the process to buffer message data and they are typically pinned to physical memory which makes them a scarce resource that must be allocated across all processes. In the base-level U-Net architecture send and receive queues hold descriptors with information about the destination, respectively origin, endpoint addresses of messages, their length, as well as offsets within the communication segment to the buffers holding the data. Free queues hold descriptors for free buffers that are made available to the network interface for storing arriving messages.

As an optimization for small messages—which are used heavily as control messages in protocol implementation—the send and receive queues may hold entire small messages in descriptors (i.e., instead of pointers to the data). This avoids buffer management overheads and can improve the round-trip latency dramatically. The size of these small messages is implementation dependent and typically reflects the properties of the underlying network.

The management of send buffers is entirely up to the process: the U-Net architecture does not place any constraints on the size or number of buffers nor on the allocation policy used. The only restrictions are that buffers lie within the communication segment, that they be properly aligned for the requirements of the network interface (e.g., to allow DMA transfers), and that each message be spread over not more than a small fixed number of buffers. The process also 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.

3.6 Kernel emulation of U-Net

Given that communication segments and message queues generally are scarce resources, it is often impractical to provide every process with U-Net endpoints and furthermore many applications (such as telnet) do not really benefit from that level of performance. Yet, for software engineering reasons it may well be desirable to use a single interface to the network across all applications. The solution to this dilemma is to provide applications with kernel-emulated U-Net endpoints. To the application these emulated endpoints look just like regular ones, except that the performance characteristics are quite different because the kernel multiplexes all of them onto a single real endpoint.

3.7 Direct-Access U-Net architecture

Direct-access U-Net is a strict superset of the base-level architecture. It allows communication segments to span the entire address space of a process and it allows senders to specify an offset in the destination communication segment at which the message data is to be deposited.

The main advantage of the direct-access architecture over the base-level is that message data can be transferred directly into application data structures without any intermediate copy into a buffer. While this form of communication requires quite some synchronization between communicating processes, parallel language implementations, such as Split-C, can easily take advantage of this facility.

The main problem with the direct-access U-Net architecture is that it is difficult to implement on current workstation hardware: the NI must essentially contain an MMU that is kept consistent with the main processor's and the NI must be able to handle incoming messages which are destined to an unmapped virtual memory page. At a more basic hardware level, the limited number of address lines on most I/O buses makes it impossible for an NI to access all of physical memory such that even with an on-board MMU it is very difficult to support arbitrary-sized communication segments.

4 U-Net on a first-generation ATM interface

The Fore systems SBA-100 ATM interface is typical of the first generation of ATM interfaces available. It is extremely simple and rather similar to the network interfaces used in parallel machines, such as the CM-5. The SBA-100 provides a 36-cell deep output FIFO as well as a 292-cell input FIFO. To send a cell the processor stores 56 bytes into the memory-mapped output FIFO and to receive a cell it reads 56 bytes from the input FIFO. A register in the interface indicates the number of cells available in the input FIFO. The only function performed in hardware, beyond simply moving cells onto/off the fiber, is ATM header CRC calculation. In particular, no DMA, no payload CRC calculation[1], and no segmentation and reassembly of multi-cell packets are supported by the interface.

4.1 U-Net/100 implementation

SBA-100 does not have any protection mechanisms in hardware which would allow mapping the device into user-space, nor can it be programmed to implement the U-Net architecture directly. The U-Net architecture is therefore implemented by the main processor and the

1. The card calculates the AAL3/4 checksum over the payload but not the AAL5 CRC required here.

kernel provides emulated U-Net endpoints to the applications as described in §3.6.

The implementation uses previously developed technology[17] consisting of two parts: a device driver that is dynamically loaded into the kernel and a user-level library implementing the AAL5 segmentation and reassembly (SAR) layer. A fast transmission path is implemented, consisting of two trap instructions which lead directly to code for sending and receiving individual ATM cells. The traps to send and receive cells are carefully crafted assembly language routines. Each routine is small (28 and 43 instructions for the send and receive traps, respectively).

For each endpoint U-Net allocates a corresponding communication segment and queues. Connections must be opened explicitly and a single page is assigned to each one for placing outgoing cells and a number of pages are held for receiving messages.

The AAL5 SAR library implements the base-level U-Net interface; it handles segmentation and reassembly of PDUs and checks the CRC. The AAL5 library breaks the PDU into ATM cells, calculates the payload CRC in the process, writes the cells into the send segment, and calls the write trap to transfer the PDU into the SBA-100 transmit FIFO.

Reception of messages takes place when the application calls the receive routine of the AAL5 layer as part of its poll routine. AAL5 receives cells by calling the read trap, reassembling the cells into a PDU and checking the payload CRC of the received message, until there are no more cells left to pick up from the NI. Small (up to 40 bytes) messages are placed in an entry in the receive queue. Larger messages are placed in the receive segment at addresses obtained from the free queue, and the offsets are written into the receive queue entry. U-Net also provides the address of the originating endpoint and the PDU size. Receive buffers are returned to the free queue by the application.

4.2 Performance

The U-Net implementation was evaluated on two 60Mhz SPARCstation-20s running SunOS 4.1.3 and equipped with Fore Systems SBA-100 interfaces. The ATM network consists of 140Mbit/s TAXI fibers leading to a Fore Systems ASX-200 switch

The end-to-end round trip time of a single-cell message, measured at the U-Net/100 interface, is 66µs. A consequence of the lack of hardware to compute the AAL5 CRC is that 33% of the send overhead and 40% of the receive overhead in the AAL5 processing is due to CRC computation. The cost breakup is shown in Table 1. Given the send and receive overheads, the U-Net/100 provides a bandwidth of 6.8MBytes/s for PDUs of 1KBytes.

Operation	Time (µs)
1-way send and rcv across switch (at trap level)	21
Send overhead (AAL5)	7
Receive overhead (AAL5)	5
Total (one-way)	33

Table 1: Cost breakup for a single-cell round-trip (AAL5)

5 U-Net on a second-generation ATM interface

The second generation of ATM network interfaces produced by Fore Systems, the SBA-200, is substantially more sophisticated than the SBA-100 and includes an on-board processor to accelerate segmentation and reassembly of PDUs as well as to transfer data to/from host memory using DMA. This processor is controlled by firmware which is downloaded into the on-board RAM by the host. The U-Net implementation described here uses this feature to implement the base-level architecture directly on the SBA-200.

The SBA-200 consists of an Intel i960 processor, 256Kbytes of memory, a DMA-capable SBus interface, a simple FIFO interface to the ATM fiber (similar to the SBA-100), and an AAL5 CRC generator. The i960 is clocked at 25Mhz and the DMA has a “fly-by” feature such that cell data need not pass through the i960’s registers as it moves between the SBus and the network FIFOs. The host processor can map the SBA-200 memory into its address space in order to communicate with the i960 during operation.

The experimental set-up used consists of five 60Mhz Sparcstation-20 and three 50Mhz Sparcstation-10 workstations connected to a Fore Systems ASX-200 ATM switch with 140Mbit/s TAXI fiber links.

5.1 Fore firmware operation and performance

The complete redesign of the SBA-200 firmware for the U-Net implementation was motivated by an analysis of Fore’s original firmware which showed poor performance. The apparent rationale underlying the design of Fore’s firmware is to off-load the specifics of the ATM adaptation layer processing from the host processor as much as possible. The kernel-firmware interface is patterned after the data structures used for managing BSD mbufs and System V streams bufs. It allows the i960 to traverse these data structures using DMA in order to determine the location of message data, and then to move it into or out of the network rather autonomously.

The performance potential of Fore's firmware was evaluated using a test program which maps the kernel-firmware interface data structures into user space and manipulates them directly to send raw AAL5 PDUs over the network. The measured round-trip time was approximately 160µs while the maximum bandwidth achieved using 4Kbyte PDUs was 13Mbytes/sec. This performance is rather discouraging: the round-trip time is almost 3 times larger than using the much simpler and cheaper SBA-100 interface, and the bandwidth for reasonable sized PDUs falls short of the 15.2Mbytes/sec peak fiber bandwidth.

A more detailed analysis showed that the poor performance can mainly be attributed to the complexity of the kernel-firmware interface. The message data structures are more complex than necessary and having the i960 follow linked data structures on the host using DMA incurs high latencies. Finally, the host processor is much faster than the i960 and so off-loading can easily backfire.

5.2 U-Net firmware

The base-level U-Net implementation for the SBA-200 modifies the firmware to add a new U-Net compatible interface[1]. The main design considerations for the new firmware were to virtualize the host-i960 interface such that multiple user processes can communicate with the i960 concurrently, and to minimize the number of host and i960 accesses across the SBus.

The new host-i960 interface reflects the base-level U-Net architecture directly. Communication segments are pinned to physical memory and mapped into the i960's DMA space, receive queues are similarly allocated such that the host can poll them without crossing the Sbus, while send and free queues are actually placed in SBA-200 memory and mapped into user-space such that the i960 can poll these queues without DMA transfers.

The i960 provides protection to user processes on a per endpoint basis. Every endpoint to endpoint connection is associated with a transmit/receive VCI pair[2] which is registered with the i960 via the kernel at the time of connection set up. For such control operations, there is a single i960-resident command queue that is used by the kernel. Processes can only access the command queue through a system call to the device driver that interfaces with the SBA-200 and the kernel can validate the connection request at set up time. In the current prototype, the connection set up checks have not been implemented. The communication segments and message queues for distinct endpoints are disjoint and are only present in the address space of the process that creates the endpoint.

1. For software engineering reasons, the new firmware's functionality is a strict superset of Fore's such that the traditional networking layers can still function while new applications can use the faster U-Net.
2. ATM is a connection-oriented network that uses virtual circuit identifiers (VCIs) to name one-way connections.

In order to send a PDU, the host uses a double word store to the i960-resident transmit queue to provide a pointer to a transmit buffer, the length of the PDU and the destination endpoint address to the i960. Single cell PDU sends are optimized as a special case because many small messages are less than a cell in size. For larger sized messages, the host-i960 DMA occurs in 1K byte chunks and uses the "fly-by" to minimize transfer times and to compute the AAL5 CRC. The entire transmission process is somewhat pipelined where the i960 requests a 1Kbyte chunk at one go and keeps putting the data on the network as it appears in the DMA input FIFO instead of requesting one cell payload at a time and waiting for the read to complete before sending the cell out.

To receive cells from the network, the i960 periodically polls the network input FIFO. Receiving single cell messages is special-cased to improve the round-trip latency for small messages. The single cell messages are directly transferred into the next receive queue entry which is large enough to hold single cell messages—this avoids buffer allocation and extra DMA for the buffer pointers. Longer messages are transferred to fixed size receive buffers whose offsets in the communication segment are pulled off the i960-resident free queue. When the last cell of the PDU is received, the message descriptor including the pointers to the buffers is DMA-ed into the next receive queue entry.

5.3 Performance

Figure 4 shows the round trip times for messages up to 1K bytes on the raw base level U-Net implementation over the SBA-200, i.e. the time for a message to go from one host to another via the switch and back. The round-trip time is 60µs for a one-cell message due to the optimization, which is rather low, but not quite at par with parallel machines, like the CM-5, where custom network interfaces allow round-trips in 12µs. Longer messages start at 86µs for 41 bytes and cost roughly and extra Xµs per additional 48 bytes. Figure 5 shows the bandwidth over the raw base level U-Net interface in Mbytes/sec for message sizes varying from 50 to 8K bytes. It is clear from the graph that with PDU sizes as low as 1Kbytes, the fiber can be saturated.

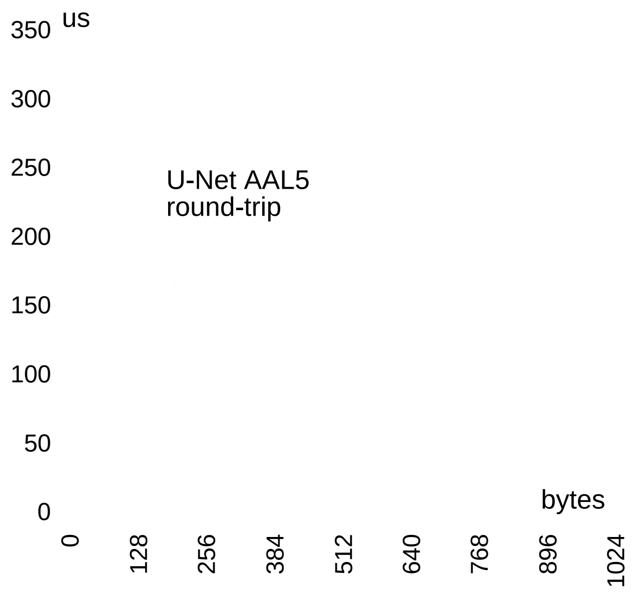

Figure 4: Round-trip times over the raw U-Net AAL5 interface as a function of message size

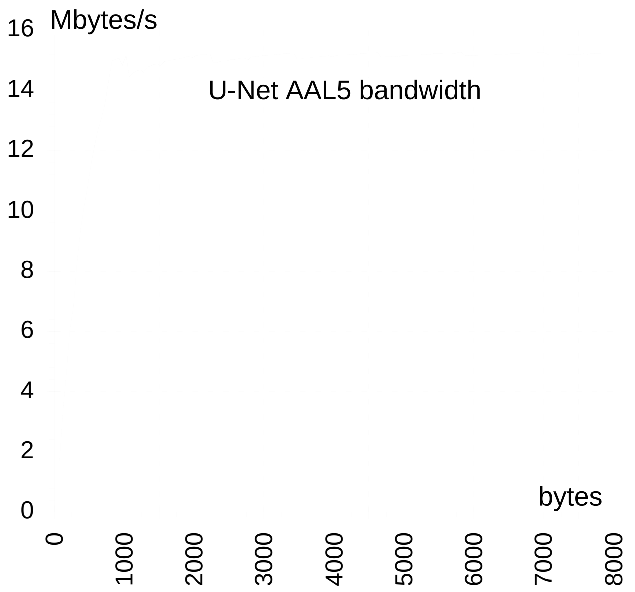

Figure 5: Bandwidth over the raw U-Net AAL5 interface as a function of message size.

5.4 Memory requirements

The current implementation uses a fixed number of memory pages pinned down to physical memory as communication segments for all endpoints. These pages are also mapped to the SBA-200's DMA space. In addition, each endpoint has its own set of send, receive and free buffer queues, out of which two reside on the i960 and are mapped to user-space. The number of distinct applications that can be run concurrently is therefore limited by the amount of memory that can be pinned down on the host, the size of the DMA address space and, the i960 memory size. Memory resource management is therefore an important issue if access to the network interface is to be scalable. A reasonable approach would be to provide a mechanism by which the i960, in conjunction with the kernel, would provide some elementary memory management functions which would allow dynamic allocation of the DMA address space to the communication segments of active user processes. The exact mechanism to achieve such an objective without compromising the efficiency and simplicity of the interface remains a challenging problem.

6 TCP/IP and UDP/IP protocols.

The success of new abstractions often depends on the level to which they are able to support legacy systems. In modern distributed systems this comes down to the need to support the TCP/IP protocol suite augmented with RPC and group oriented communication. U-Net provides a number of these protocols as a user-level library built on the base-level U-Net functionality.

Performance of a new networking technology is often put into perspective by measuring the throughput and latency of the TCP & UDP protocols. The performance of these protocols using the vendor supplied ATM driver software is disappointing; the UDP maximum bandwidth is only achieved by using very large transfer sizes (larger than 8Kbytes), while TCP under all circumstances will not perform better that at 55% of the maximum. The latency performance, however, is even more dramatic: for small message sizes the latency of both UDP and TCP messages is worse than the latency on Ethernet: it simply does not reflect the increased capabilities of the ATM technology. Figure 6 shows the latency of the Fore-ATM based protocols compared to those over Ethernet.

6.1 The non-problem with TCP/IP

The TCP/IP suite of protocols is often considered to be ill-suited for use over high-speed networks like ATM, but experience has shown that often the core of the problems with TCP/IP lie in the particular *implementations* and their *integration* into the operating system. This is indeed the case where the Fore driver software tries to deal with the generic, low-performance buffer strategies of the BSD based kernel. In contrast, using U-Net, TCP and UDP can be implemented at user-level which allows the implementation to be tuned to the characteristics of the network and without the need to generalize. Specifically, U-Net TCP and UDP tune the buffer and timer management and allow better error and congestion feed-back to the application. As a result, U-Net TCP and UDP deliver the low-latency and high bandwidth communication expected of ATM networks without resorting to excessive buffer schemes or the use of large network transfer units, while maintaining full interoperability with non-U-Net implementations.[1]

6.2 Removing the TCP/IP kernel resource problems.

A major problem in the implementation of kernel based protocols is the limited kernel resources available which need to be shared between many potential network-active processes. By implementing TCP & UDP at user-level, efficient solutions are available for problems which are caused by using the kernel as the single protocol processing unit. Not only does U-Net remove all copy operations from the protocol path but it also allows the buffering strategy to depend on the resources at the process instead of the scarce kernel network buffers. The restricted size of the receive socket buffer (max. 52Kbytes) has been a common problem with the BSD kernel communication path: already at Ethernet speeds buffer overrun is the cause of message loss in the case of high bandwidth UDP data streams. By removing this restriction, the resources of the actual recipient, instead of those of the intermediate processing unit, now become the main control factor and this can efficiently be incorporated into the end-to-end flow-control mechanisms.

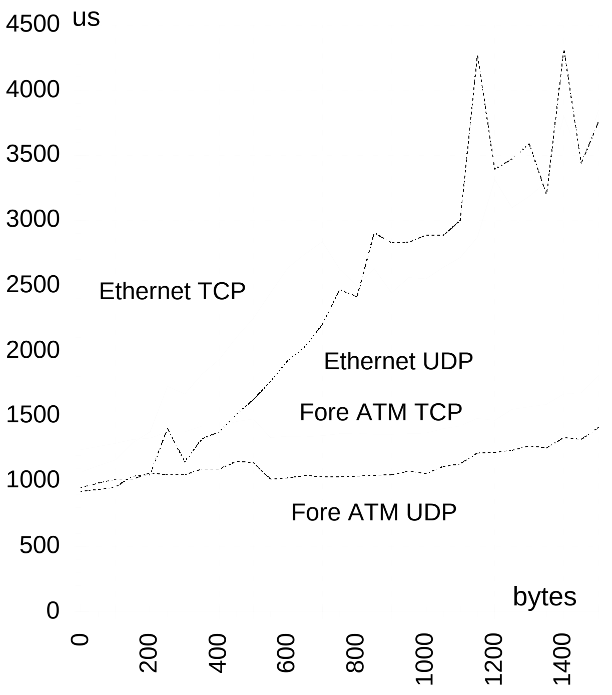

Figure 6: TCP and UDP round-trip latencies over ATM and Ethernet. as a function of message size.

1. U-Net TCP and UDP are interoperable in the sense that they comply with the standard TCP/IP and UDP/IP RFC's. The use of ATM VCIs however is currently incompatible with Fore's implementations.

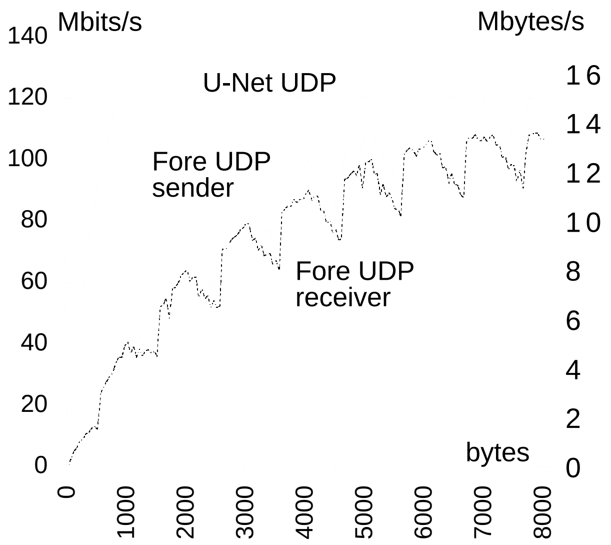

Figure 7: UDP bandwidth as a function of message size.

The impact of the kernel buffer management in combination with the Fore buffering scheme is seen in Figure 7 which shows the UDP throughput. The saw-tooth behavior is caused by the buffer allocation scheme, where first large (1Kb) buffers are filled with data and the remainder (if less than 512 bytes) is copied into single small mbufs (112 bytes). This allocation method has a strong degrading effect on the performance of the protocols because unlike the large cluster buffers that have a reference count mechanism associated with them, the smaller mbufs do not have this optimization.

For U-Net, a scatter-gather message mechanism is implemented to support efficient construction of network buffers. The data blocks are allocated within the receive and transmit communication segments and a simple reference count mechanism allows them to be shared by several messages without the need for copy operations.

Given that a process has full control over sending U-Net messages it is possible to provide correct feedback to the application about the state of the transmission queue and it is easy to establish a back-pressure mechanism when the transmission queues are full. This overcomes, for example, problems with the current SunOS implementation which will drop random packets from the device transmit queue if there is overload, all without notifying the sending process.

6.3 IP

The U-Net/IP implementation exploits functionality offered by the U-Net architecture to select the protocol module that handles each message. The demultiplex information tagged to the message by U-Net,. The functionality of IP on the outgoing path is reduced to mapping every message onto a particular connection. IP over U-Net exports an MTU of 9Kbytes and does not support fragmentation, as this is known to be a potential source for wasting bandwidth and triggering packet retransmissions[10]. TCP provides its own fragmentation mechanism and the application level should assist UDP in achieving the same functionality. Given this strongly reduced functionality IP has been collapsed into the transport level protocols to allow efficient processing.

6.4 UDP

The core functionality of UDP is twofold: an additional layer of demultiplexing over IP based on port identifiers and some protection against corruption by adding a 16 bit checksum on the data and header parts of the message. In the U-Net implementation the demultiplexing is simplified by using the source communication segment information passed-on by U-Net. The checksum adds a processing overhead of 1µs per 100 bytes, and it can be switched off by applications that use data protection at a higher level or are satisfied by the 32-bit CRC at the U-Net AAL5 level.

The performance of U-Net UDP is compared to the kernel UDP in Figures 7 and 8. The first shows the achieved bandwidth while the latter plots the end-to-end round-trip latency as a function of message size For the kernel UDP the bandwidth is measured as perceived at the sender and as actually received: the losses can all be attributed to kernel buffering problems at both sending

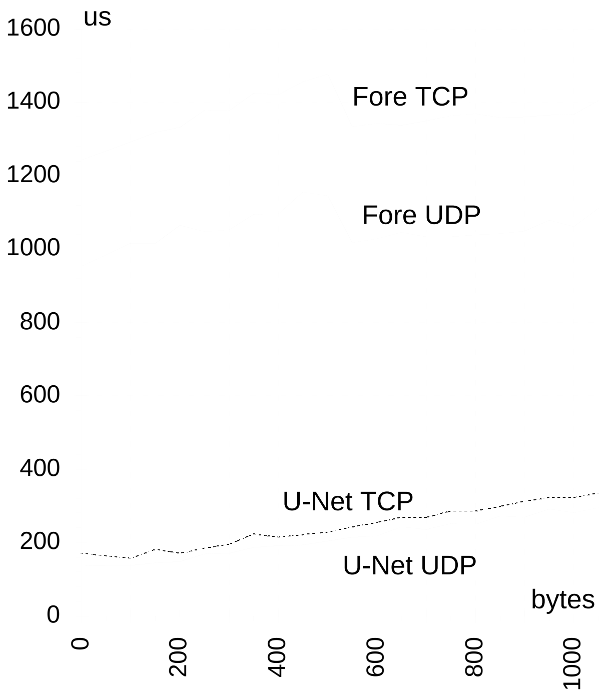

Figure 8: UDP and TCP round-trip latencies as a function of message size.

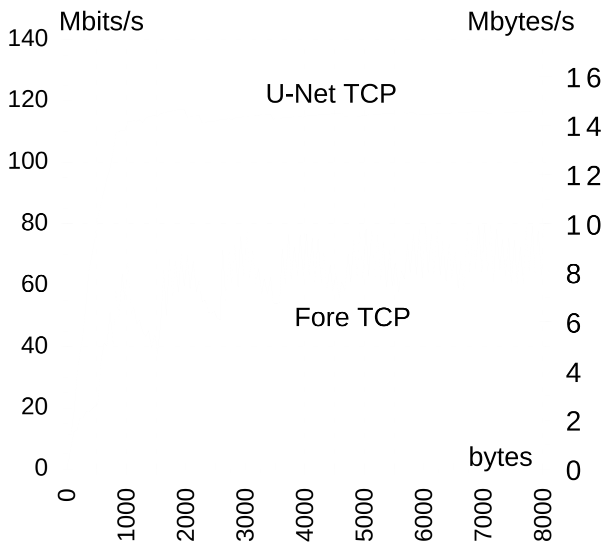

Figure 9: TCP bandwidth as a function of data generation by the application.

and receiving hosts. U-Net UDP does not experience any losses and only the receive bandwidth is shown.

6.5 TCP

TCP adds two properties that make it an attractive protocol to use in a number of settings: reliability and flow control. Reliability is achieved through a simple acknowledgment scheme and flow control through the use of advertised receive windows. TCP over high-speed networks has been studied extensively, especially in a wide-area setting[14] and a number of changes and extensions have been proposed to make TCP function correctly in settings where a relatively high delay can be expected[3]. It has been argued lately that these changes are also needed to solve the deficiencies that occur because of the high-latency of the ATM kernel software. U-Net TCP is able to achieve acceptable performance without any modifications to the general algorithms, without the use of large sequence numbers and without extensive buffer reservations.

The performance of TCP does not depend as much on the rate with which the data can be pushed out on the network as on the product of bandwidth and round-trip time, which indicates the amount of buffer space needed to maintain a steady reliable high speed flow. The window size indicates how many bytes the module can send before it has to wait for acknowledgments and window updates from the receiver. If the updates can be returned to the sender in a very timely manner only a relatively small window is needed to achieve the maximum bandwidth. Figure 9 shows that in most cases U-Net TCP achieves a 14-15 Mbytes/sec bandwidth using an 8Kbyte window, while the kernel TCP/ATM combination will, even in the case of the maximum 64K window, not achieve a higher bandwidth than 7 Mbytes/sec. The round-trip latency performance of both kernel and U-Net TCP implementations is shown in Figure 8 and highlights the fast U-Net TCP round-trip which permits the use of a small window.

Another important factor is the size of the segments that are transmitted; using larger segments it is more likely that the maximum bandwidth can be achieved, but recent work has shown that TCP can perform poorly over ATM if the segment size is large[10] due to the fact that the underlying cell reassembly mechanism causes the entire segment to be discarded if a single ATM cell is dropped. A number of solutions are available, but none provide a mandate to use large segment sizes. The standard configuration for U-Net TCP uses 2048 byte segments, which is sufficient to achieve 13-14 Mbytes/sec bandwidth in combination with an 8 Kbyte window.

Another potential problem that has been solved within U-Net TCP is the bad ratio between the granularity of the protocol timers and the round-trip time estimates. The retransmission timer in TCP is set as a function of the estimated round trip time, which is in the range from 60 to 700 microseconds, but the kernel protocol timer (*pr_slow_timeout*) has a granularity of 500 milliseconds. When a TCP packet is discarded because of cell loss or dropped due to congestion the retransmit timer gets set to a relatively large value, compared to the actual round-trip time. To ensure timely reaction to possible packet loss U-Net TCP uses a 1 millisecond timer granularity, which is constrained by the reliability of the Unix user-level interval timer.

The BSD implementation uses another timer (*pr_fast_timeout*) for the transmission of a delayed acknowledgment in the case that no send data is available for piggybacking and that a potential transmission deadlock needs to be resolved. This timer is used to delay the acknowledgment of every second packet for up to 200ms. U-Net TCP does not use this delayed ack strategy given the low cost of an active acknowledgment: which consists of only a 40 byte TCP/IP header and thus can be handled efficiently by inline U-Net reception. As a result, the available send window is updated in the most timely manner possible, laying the foundation for maximal bandwidth exploitation.

7 U-Net Active Messages implementation and performance

The U-Net Active Messages (UNAM) layer is a prototype that conforms to the Generic Active Messages (GAM) 1.1 specification[6]. Active Messages is a mechanism that allows efficient overlapping of communication with computation in multiprocessors. Communication using Active Messages is in the form of requests and matching replies. An active message contains the address of a handler that gets called on receipt of the message followed by upto four words of arguments. The function of the handler is to pull the message out of the network and integrate it into the ongoing computation. A request message handler may or may not send a reply message. However, in order to prevent livelock, a reply message handler cannot send another reply.

Generic Active Messages consists of a set of primitives that higher level layers can use to initialize the GAM interface, send request and reply messages and perform bulk gets and stores. GAM provides a guarantee of best effort message delivery which implies that a message that is sent will be delivered to the recipient barring network partitions, node crashes, or other catastrophic failures.

7.1 Active Messages implementation

The UNAM implementation consists of a user level library that exports the GAM 1.1 interface and uses the U-Net interface. The library is rather simple and only performs the flow-control and retransmissions necessary to implement best-effort delivery and the Active Messages-specific part is just dispatching handlers.

7.1.1 Flow Control Issues

In order to ensure reliable message delivery, UNAM uses a simple window-based flow control protocol. The window size w is fixed and every outgoing PDU is assigned a sequence number in the range $[0, 2w - 1]$. Every endpoint preallocates enough transmit and receive buffers to be able to hold two full windows of received as well as transmitted messages for every endpoint it communicates with, one window each for requests and replies, respectively.

Request messages which do not require a reply are explicitly acknowledged. The distinction between request and reply messages allows several acknowledgments to be piggy-backed onto the same reply message which reduces the network traffic. A standard retransmission mechanism is used to deal with lost requests or replies. It should be noted that the flow control implemented here is an end-to-end flow control mechanism which does not attempt to minimize message losses due to congestion in the network.

7.1.2 Sending and Receiving U-Net Active Messages

To send a request message, UNAM first processes any outstanding messages in the receive queue, drops a copy of the message to be sent into a pre-allocated transmit buffer and pushes a descriptor onto the send queue. If

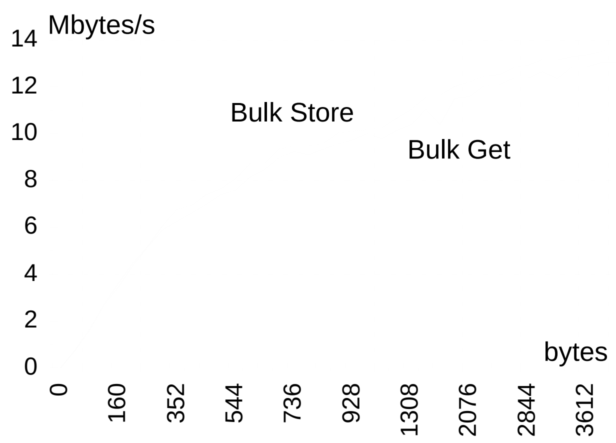

Figure 10: Bulk Store and Get bandwidths for UNAM

the send window is full, the sender polls for messages received until there is space in the send window or until a time-out occurs and the entire window of messages is retransmitted. The sending of reply messages or explicit acknowledgment is similar except that the sender does not poll for messages before sending (in order to avoid live-lock).

The UNAM layer receives messages by explicit polling. On message arrival, UNAM loops through the receive queue, pulls the messages out of the receive buffers, dispatches the handlers, sends explicit acknowledgments where necessary, and frees the buffers and the receive queue entries.

7.2 U-Net Active Messages micro-benchmarks

Three different micro-benchmarks were run to determine the round trip time for single cell messages, the bandwidth for bulk store and the bandwidth for pipelined bulk get operations. The round trip time was estimated by repeatedly sending a single cell active message to a remote host specifying a handler that simply replies. The measured round trip time is 66µs. Since the round trip time on the raw base level U-Net is 60µs the UNAM overhead is about 6µs to send a message, receive it, reply and receive the reply.

The bulk store bandwidth was measured by repeatedly storing a block of a specified size to a remote node in a loop and measuring the total time taken. The measurement was repeated for sizes varying from 16 to 3800 bytes. The pipelined bulk get bandwidth was measured similarly by repeatedly sending bulk get requests for a specified size in a loop and waiting for all the bulk gets to complete. This measurement was also repeated for sizes varying from 16 to 3800 bytes. Figure 10 shows the store and get bandwidths for the varying message sizes. The upper curve represents the store bandwidth while the lower curve represents the get bandwidth. The maximal store bandwidth achieved is 13.66 Mbytes/sec for transfer sizes of 3800 bytes which still leaves room for improvement.

The pipelined get bandwidth closely follows the store bandwidth for small sized messages but begins to fall behind around a transfer size of 400 bytes and goes up to a maximum of 13.04 Mbytes/sec for transfer sizes of 3800 bytes.

7.3 Summary

[...]

8 Split-C application benchmarks

Split-C[4] is a simple parallel extension to C for programming distributed memory machines using a global address space abstraction. It is implemented on top of Generic Active Messages and is used here to demonstrate the impact of U-Net on applications written in a parallel language. A Split-C program is comprised of a thread of control per processor from a single code image and the threads interact through reads and writes on shared data. The type system distinguishes between local and global pointers such that the compiler can issue the appropriate calls to Active Messages whenever a global pointer is dereferenced. Thus, dereferencing a global pointer to a scalar variable turns into a request and reply Active Messages sequence exchange with the processor holding the data value. Split-C also provides bulk transfers which map into Active Message bulk gets and stores to amortize the overhead over a large data transfer.

Split-C has been implemented on the CM-5, Paragon, SP-1, Meiko CS-2, and Cray T3D supercomputers as well as over the U-Net Generic Active Messages. A small set of application benchmarks is used here to compare the U-Net version of Split-C to the CM-5[18] and Meiko CS-2[15] versions. This comparison is particularly interesting as the CM-5 and Meiko machines are easily characterized with respect to the U-Net ATM cluster as shown in Table 2: the CM-5's processors are

Machine	CPU speed	message overhead	round-trip latency	network bandwidth
CM-5	33 Mhz Sparc-2	3µs	12µs	10Mb/s
Meiko CS-2	40Mhz Supersparc	11µs	25µs	39Mb/s
U-Net ATM	50/60 Mhz Supersparc	6µs	66µs	14Mb/s

Table 2: Comparison of CM-5, Meiko CS-2, and U-Net ATM cluster computation and communication performance characteristics

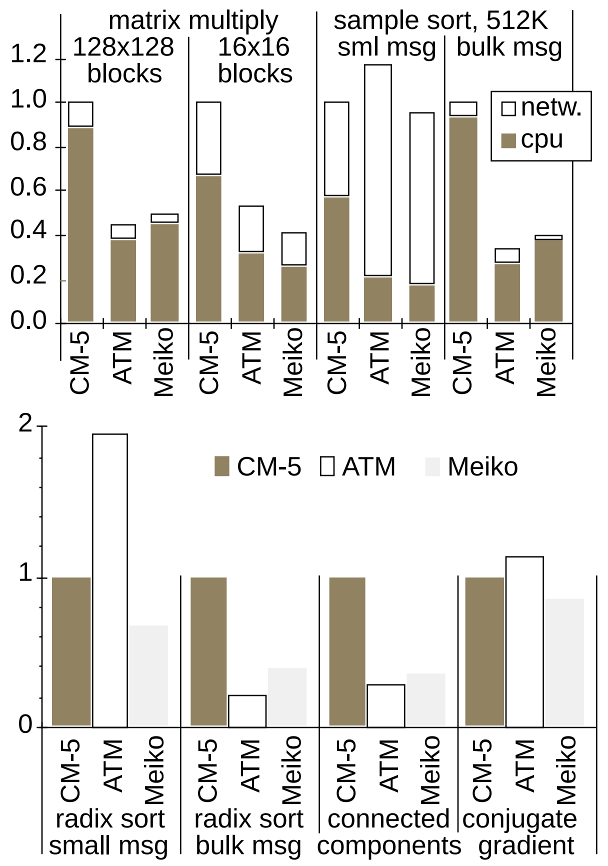

Figure 11: Comparison of seven Split-C benchmarks on the CM-5, the U-Net ATM cluster, and the Meiko CS-2. The execution times are normalized to the CM-5 and the computation/communication breakdown is shown for three

slower than the Meiko's and the ATM cluster's, but its network has lower overheads and latencies. The CS-2 and the ATM cluster have very similar characteristics with a slight CPU edge for the cluster and a faster network for the CS-2.

The Split-C benchmark set used here is comprised of seven programs: a blocked matrix multiply[4], a sample sort optimized for small messages[5], the same sort optimized to use bulk transfers[15], two radix sorts optimized for small and bulk transfers, respectively, a connected components algorithm[11], and a conjugate gradient solver. The matrix multiply and the sample sorts have been instrumented to account for time spent in local computation phases and in communication phases separately such that the time spent in each can be related to the processor and network performance of the machines. (The other benchmarks will be instrumented similarly for the final paper.) The execution times for runs on eight processors are shown in Figure 11; the times are normalized to the total execution time on the CM-5 for ease of comparison. The matrix multiply uses matrices of 4 by 4 blocks with 128 by 128 double floats each. The main loop multiplies two blocks while it prefetches the two blocks needed in the next iteration. The results show clearly the CPU and network bandwidth disadvantages of the CM-5. The sample sort sorts an array of 4 million 32-bit integers with arbitrary distribution. The algorithm first picks 64 samples on each processor, then sorts all the samples on one processor, selects splitters to determine which range of values should end up on each processor, broadcasts the splitters, permutes all the values to the right processor (this is the main communication phase), and finally each processor sorts its values locally (which contributes most to the computation time). The version optimized for small messages packs two values per message while the one optimized for bulk transfers presorts the local values such that each processor sends exactly one message to every other processor during the permutation phase. The performance again shows the CPU disadvantage of the CM-5 and in the small message version that machine's per-message overhead advantage. The ATM cluster and the Meiko come out roughly equal with a slight CPU edge for the ATM cluster and a slight network bandwidth edge for the Meiko. The bulk message version improves the Meiko and ATM cluster performance dramatically with respect to the CM-5 which has a lower bulk-transfer bandwidth. The performance of the radix sort and the connected components benchmarks further demonstrate that the U-Net ATM cluster of workstations is roughly equivalent to the Meiko CS-2 and performs worse than the CM-5 in applications using small messages (such as the small message radix sort and connected components) but better in ones optimized for bulk transfers.

9 Summary

The main objective of U-Net, to provide high-performance low-latency communication, has been accomplished: The processing overhead on messages has been minimized so that the latency experienced by the application is dominated by the actual message transmission time.

Using U-Net the round-trip latency for messages smaller than 40 bytes is about 60 µsec. This compares favorably to other recent research results: the *application device channels* (U. of Arizona) achieve 150 µsec latency for single byte messages and 16 byte messages in the HP Jetstream environment have latencies starting at 300 µsec. Both research efforts however use dedicated hardware capable of over 600 Mbits/sec compared to the 140 Mbits/sec standard hardware used for U-Net.

Although the main goal of the U-Net architecture was to remove the processing overhead to achieve low-latency, a secondary goal, namely the delivery of maximum network bandwidth, even with small messages, has also been achieved. With message sizes as small as 900 bytes the network is saturated, while at smaller sizes the dominant bottleneck is the i960 processor on the network interface.

U-Net also demonstrates that removing the kernel from the communication path can offer more than just high performance: U-Net presents a simple network interface architecture which simultaneously supports traditional inter-networking protocols as well as novel communication abstractions like Active Messages. The TCP and UDP protocols implemented using U-Net achieve latencies and throughput close to the raw maximum and Active Messages round-trip times are only a few microseconds over the absolute minimum.

The final comparison of the 8-node ATM cluster with the Meiko CS-2 and TMC CM-5 supercomputers using a small set of Split-C benchmarks demonstrates that with the right communication substrate networks of workstations can indeed rival these specially-designed machines. This encouraging result should, however, not obscure the fact that significant additional system resources, such as parallel process schedulers and parallel file systems, still need to be developed before the cluster of workstations can be viewed as a unified resource.

10 Acknowledgments

U-Net wouldn't look the way it does without the numerous discussions, the many email exchanges, and the taste of competition we had with friends in the UC Berkeley NoW project, in particular David Culler, Alan Mainwaring, Rich Martin, and Lok Tin Liu.

The Split-C section was only possible thanks to the generous help of Klaus Eric Schauser at UC Santa Barbara who shared his Split-C programs and provided quick access to the Meiko CS-2 which is funded under NSF grant #. The CM-5 results were obtained on the UCB machine, funded under NSF grant #. Thanks also to the UCB Split-C group for the benchmarks, in particular Arvind Krishnamurthy.

Most of the ATM workstation cluster was purchased under contract F30602-94-C-0224 from Rome Laboratory, Air Force Material Command. We also thank Fore Systems for making the source of the SBA-200 firmware available to us: without it we would have been stuck.

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

[3] D. Borman, R. Braden, and V. Jacobson. *TCP Extensions for High Performance*. RFC 1323, May 1992.

[4] 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.

[5] 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.

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

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

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

[9] 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*. In Proc. of SIGCOMM-94, pages 14-23, Aug. 1994.

[10] C. Kent and J. Mogul. Fragmentation Considered Harmful In Proc. of SIGCOMM-87. pages 390-410. Aug 1987.

[11] A. Krishnamurthy, S. Lumetta, D. E. Culler, and K. Yelick. *Connected Components on Distributed Memory Machines.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 00, 0000

[12] M. Lin, J. Hsieh, D. H. C. Du, J. P. Thomas, and J. A. MacDonald. *Distributed Network Computing over Local ATM Networks.* IEEE Journal on Selected Areas in Communications, Special Issue on ATM LANs, to appear, 1995.

[13] C. Maeda and B.N. Bershad. *Protocol Service Decomposition for High-Performance Networking.* In Proc. of the 14th SOSP, pages 244-255. Dec. 1993.

[14] A. Romanow and S. Floyd. *Dynamics of TCP traffic over ATM networks*. In Proc. of SIGCOMM-94. pages 79-88, Aug. 94.

[15] K.E Schauser and C. J. Scheiman. E*xperience with Active Messages on the Meiko CS-2*. To appear.

[16] 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.

[17] T. von Eicken, Anindya Basu and Vineet Buch. *Low-Latency Communication Over ATM Networks Using Active Messages.* IEEE Micro, Feb. 1995, pages 46-53.

[18] 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.