Proc. of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, December 3-6, 1995

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

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

Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract

The U-Net communication architecture provides processes with a virtual view of a network interface 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 Messages can be implemented efficiently. A U-Net prototype on an 8-node ATM cluster of standard workstations offers 65 microseconds round-trip latency and 15 Mbytes/sec bandwidth. It achieves TCP performance at maximum network bandwidth 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.

Many new application domains could benefit not only from higher network performance but also from a more flexible interface to the network. By placing all protocol processing into the kernel the traditional networking architecture cannot easily support new protocols or new message send/receive interfaces. Integrating application specific information into protocol processing allows for higher efficiency and greater flexibility in protocol cost management. 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[26]. Other applications can avoid copying message data by sending straight out of data structures. Being able to accommodate such application specific knowledge into the communication protocols becomes more and more important in order to be able to efficiently utilize the network and to couple the communication and the computation effectively.

One of the most promising techniques to improve both the performance and the flexibility of 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 protocol stack should be placed 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 Thinking Machines 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 net-

Authors' email: {tve,basu,buch,vogels}@cs.cornell.edu.
Software: http://www.cs.cornell.edu/Info/Projects/U-Net/

The U-Net project is supported by the Air Force Material Contract F30602-94-C-0224 and ONR contract N00014-92-J-1866.

Copyright ©1995 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that new copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted.

This document was created with FrameMaker 4.0.4

work 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 channel set-up 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 Messages 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 communication layers.

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., it allows many hardware implementations), to describe two high-performance implementations on standard workstations (Section 4), and to evaluate its performance characteristics for communication in parallel programs (Sections 5 and 6) as well as for traditional protocols from the IP suite (Section 7). 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 Motivation and related work

The U-Net architecture focuses on reducing the processing overhead required to send and receive messages as well as on providing flexible access to the lowest layer of the network. The intent is three-fold:

- provide low-latency communication in local area settings,
- exploit the full network bandwidth even with small messages, and
- facilitate the use of novel communication protocols.

2.1 The importance of low communication latencies

The latency of communication is mainly composed of processing overhead and network latency (time-of-flight). 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. 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 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 in local area communication, however, the processing overheads dominate and the improvement in transmission time is less significant in comparison. In wide area networks the speed of light eventually becomes the dominant latency component and while reducing the overhead does not significantly affect latency it may well improve throughput.

U-Net places a strong emphasis on achieving low communication overheads because 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. High processing overheads resulting in high communication latencies prevent such protocols from being 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:

- 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[2].

Finally, 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 importance of small-message bandwidth

The communication bandwidth of a system is often measured by sending a virtually infinite stream from one node to another. While this may be representative of a few applications, the demand for high bandwidths when sending many small messages (e.g., a few hundred bytes) is increasing due to the same trends that demand low latencies. U-Net specifically targets this segment of the network traffic and attempts to provide full network bandwidth with as small messages as possible, mainly by reducing the per-message overheads.

Reducing the minimal message size at which full bandwidth can be achieved may also benefit reliable data stream protocols like TCP that 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 in local area networks will keep the buffer consumption within reason and thus make it feasible to achieve maximal bandwidth at low cost.

2.3 Communication protocol and interface flexibility

In traditional UNIX networking architectures the protocol stacks are implemented as part of the kernel. This makes it difficult to experiment with new protocols and efficiently support dedicated protocols that deal with application specific requirements. Although one could easily design these protocols to make use of a datagram primitive offered by the kernel (like UDP or raw IP), doing so efficiently without adding the new protocol to the kernel stack is not possible. The lack of support for the integration of kernel and application buffer management introduces high processing overheads which especially affect reliable protocols that need to keep data around for retransmission. In particular, without shared buffer management reference count mechanisms cannot be used to lower the copy and application/kernel transfer overheads. For example, a kernel-based implementation of a reliable transport protocol like TCP can use reference counts to prevent the network device driver from releasing network buffers that must remain available for possible retransmission. Such an optimization is not available if an application specific reliable protocol is implemented in user space and has to use UDP as transport mechanism.

By removing the communication subsystem's boundary with the application-specific protocols, new protocol design techniques, such as Application Level Framing [10,16] and Integrated Layer Processing [1,6,10], can be applied and more efficient protocols produced. Compiler assisted protocol development can achieve maximum optimization if all protocols are compiled together instead of only a small subset of application specific protocols.

In more specialized settings a tight coupling between the communication protocol and the application can yield even higher savings. For example, in a high-level language supporting a form of blocking RPC no copy need to be made in case a retransmission is required as the high-level semantics of the RPC guarantee that the source data remains unmodified until the RPC completes successfully. Thus the address of a large RPC argument may well be passed down directly to the network interface’s DMA engine.

Another example is that at the moment a process requests data from a remote node it may pre-allocate memory for the reply. When the response arrives the data can be transferred directly into its final position without the allocation of intermediate buffers or any intermediate copies.

Taking advantage of the above techniques is becoming a key element in reducing the overhead of communication and can only be done if applications have direct access to the network interface.,

2.4 Towards a new networking architecture

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. The central idea in U-Net 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 1, 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.

2.5 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 single OS-Server was responsible for all network communication. The performance achieved is roughly the same as that of a monolithic BSD system.[22]

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 [14] used in the Aurora Gigabit testbed. Other techniques that are developed for the Osiris board to reduce the processing overhead are the pathfinder multiplexor [3], which is implemented in hardware and the *fbufs* cross domain buffer management [13].

At HP Bristol a mechanism has been developed for the Jetstream LAN [15] where applications can reserve buffer pools on the Afterburner [12] board. When data arrives on a VCI associated with an application data is transferred directly into the correct pool. However, the application cannot access these buffers directly: it is always forced to go through the kernel with a copy operation to retrieve the data or provide data for sending. Only the kernel based protocols could be made aware of the buffer pools and exploit them fully.

In the parallel computing community recent machines (e.g., Thinking Machines CM-5, Meiko CS-2, IBM SP-2, Cray T3D)

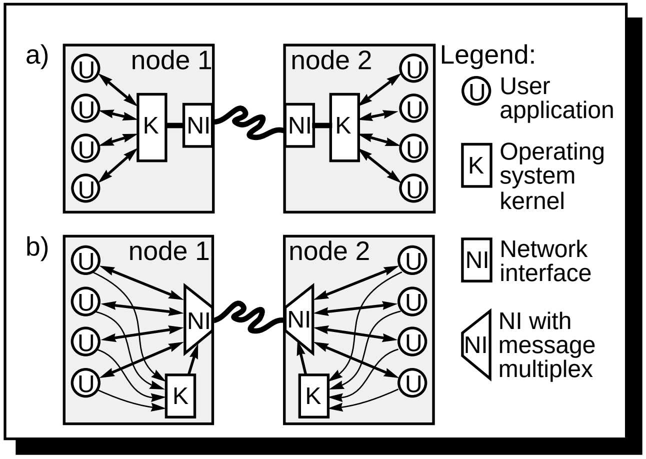

Figure 1: The traditional networking architecture (a) places the kernel in the path of all communication. The U-Net architecture (b) only uses a simple multiplexing/demultiplexing agent—that can be implemented in hardware—in the data communication path and uses the kernel only for set-up.

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[4] 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[27] 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.

2.6 U-Net design goals

Experience with network interfaces in parallel machines made it clear that providing user-level access to the network in U-Net is the best avenue towards offering communication latencies and bandwidths that are mainly limited by the network fabric and that, at the same time, offer full flexibility in protocol design and in the integration of protocol, buffering, and appropriate higher communication layers. The many efforts in developing fast implementations of TCP and other internetworking protocols clearly affirm the relevance of these protocols in high-performance networking and thus any new network interface proposal must be able to support these protocols effectively (which is typically not the case in parallel machines, for example).

The three aspects that set U-Net apart from the proposals discussed above are:

- the focus on low latency and high bandwidth using small messages,
- the emphasis on protocol design and integration flexibility, and
- the desire to meet the first two goals on widely available standard workstations using off-the-shelf communication hardware.

3 The user-level network interface architecture

The U-Net user-level network interface architecture 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 and the management of send and receive resources, such as buffers.

3.1 Sending and receiving messages

The U-Net architecture is composed of three main building blocks, shown in Figure 2: *endpoints* serve as an application's handle into the network and contain *communication segments* which are regions of memory that hold message data, and *message queues* which 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, then associates a communication segment and a set of *send*, *receive*, and *free* message queues with each endpoint.

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; this indicates that the associated send buffer can be reused.

Incoming messages are demultiplexed by U-Net based on their destination: 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, it can block waiting for the next message to arrive (using a UNIX select call), or it can register an upcall[2] with U-Net. The upcall is used by U-Net to signal that the

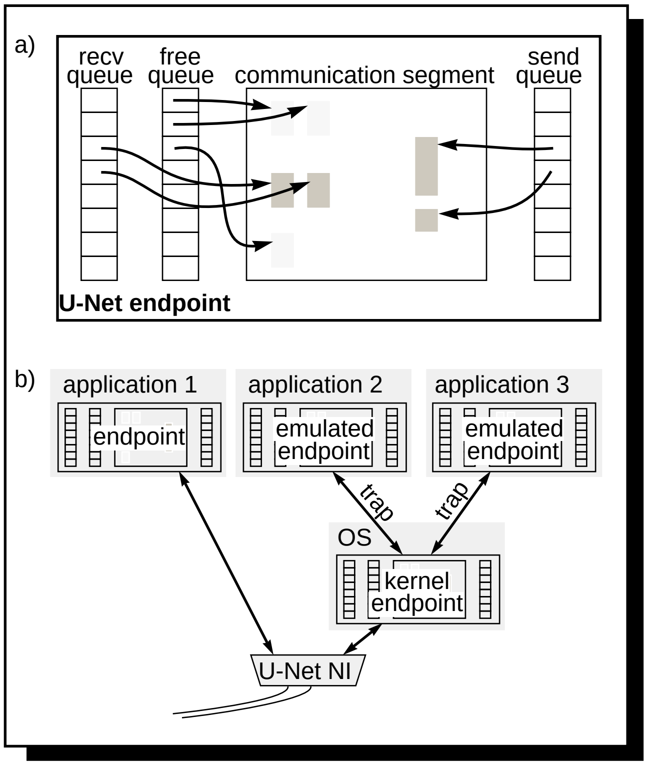

Figure 2: U-Net building blocks.
a) *Endpoints* serve as an application's handle into the network, *communication segments* are regions of memory that hold message data, and message queues *(send/recv/free queues)* hold descriptors for messages that are to be sent or that have been received.
b) Regular endpoints are serviced by the U-Net network interface directly. Emulated endpoints are serviced by the kernel and consume no additional network interface resources but cannot offer the same level of performance.

[1] The terms "process" and "application" are used interchangeably to refer to arbitrary unprivileged UNIX processes.

[2] 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.

state of the receive queue satisfies a specific condition. The two conditions currently supported are: the receive queue is non-empty and the receive queue is almost full. The first one allows event driven reception while the second allows processes to be notified before the receive queue overflows. U-Net does not specify the nature of the upcall which could be a UNIX signal handler, a thread, or a user-level interrupt handler.

In order to amortize the cost of an upcall over the reception of several messages it is important that a U-Net implementation allows 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.

3.2 Multiplexing and demultiplexing messages

U-Net uses a tag in each incoming message to determine its destination endpoint and thus the appropriate communication segment 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 ATM virtual channel identifiers (VCIs) may be used. In any case, 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.

U-Net's notion of a message tag is very similar to the idea used in parallel machines of including a parallel-process id in the header of messages. The message tag used in U-Net is more general, however, in that it allows communication between arbitrary processes, whereas a parallel-process id tag only serves communication within a parallel program running in a closed environment.

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

Endpoints and communication channels together allow U-Net to enforce protection boundaries among multiple processes accessing the network and, depending on how routes are allocated, may allow it to extend these boundaries across the network. This is achieved using two 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 only delivered to the correct destination endpoint.

Thus an application cannot interfere with the communication channels of another application on the same host. In addition, if the set-up of routes is carefully controlled by the collection of operating systems in a cluster of hosts, then this protection can be extended across the network such that no application can directly interfere with communication streams between other parties.

3.3 Zero-copy vs. true zero-copy

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 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.[1]

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 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.

3.4 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 application data structures and the network. The communication segments are allocated to buffer message data and are typically pinned to physical memory. 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 data. Free queues hold descriptors for free buffers that are made available to the network interface for storing arriving messages.

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 main restriction are that buffers lie within the communication segment and that they be properly aligned for the requirements of the network interface (e.g., to allow DMA transfers). 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.

[1] True zero copy is achieved with base-level U-Net when there is no need for the application to copy the information received to a data structure for later reference. In that case data can be accessed in the buffers and the application can take action based on this information without the need for a copy operation. A simple example of this is the reception of acknowledgment messages that are used to update some counters but do not need to be copied into longer term storage.

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 substantially. The size of these small messages is implementation dependent and typically reflects the properties of the underlying network.

3.5 Kernel emulation of U-Net

Communication segments and message queues are generally scarce resources and it is often impractical to provide every process with U-Net endpoints. 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.

3.6 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. This capability allows message data to 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 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. Thus, in essence, it requires (i) the NI to include some form of memory mapping hardware, (ii) all of (local) physical memory to be accessible from the NI, and (iii) page faults on message arrival to be handled appropriately.

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 Two U-Net implementations

The U-Net architecture has been implemented on SPARCstations running SunOS 4.1.3 and using two generations of Fore Systems ATM interfaces. The first implementation uses the Fore SBA-100 interface and is very similar to an Active Messages implementation on that same hardware described elsewhere[28]. The second implementation uses the newer Fore SBA-200 interface and reprograms the on-board i960 processor to implement U-Net directly. Both implementations transport messages in AAL5 packets and take advantage of the ATM virtual channel identifiers in that all communication between two endpoints is associated with a transmit/receive VCI pair[1].

4.1 U-Net using the SBA-100

The Fore Systems SBA-100 interface operates using programmed I/O to store cells into a 36-cell deep output FIFO and to retrieve incoming cells from a 292-cell deep input FIFO. The only function performed in hardware beyond serializing cells onto the fiber is ATM header CRC calculation. In particular, no DMA, no payload CRC calculation[2], and no segmentation and reassembly of multi-cell packets are supported by the interface. The simplicity of the hardware requires the U-Net architecture to be implemented in the kernel by providing emulated U-Net endpoints to the applications as described in §3.5.

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)

The implementation consists of a loadable device driver and a user-level library implementing the AAL5 segmentation and reassembly (SAR) layer. Fast traps into the kernel are used to send and receive individual ATM cells: each is carefully crafted in assembly language and is quite small (28 and 43 instructions for the send and receive traps, respectively).

The implementation was evaluated on two 60Mhz SPARCstation-20s running SunOS 4.1.3 and equipped with 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 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 bandwidth is limited to 6.8MBytes/s for packets of 1KBytes.

4.2 U-Net using the SBA-200

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 packets 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 custom firmware to implement the base-level architecture directly on the SBA-200.

The SBA-200 consists of a 25Mhz Intel i960 processor, 256Kbytes of memory, a DMA-capable I/O bus (Sbus) interface, a simple FIFO interface to the ATM fiber (similar to the SBA-100), and an AAL5 CRC generator. 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.

4.2.1 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

[1] ATM is a connection-oriented network that uses virtual channel identifiers (VCIs) to name one-way connections.

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

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 packets 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 packets 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.

4.2.2 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 I/O bus.

The new host-i960 interface reflects the base-level U-Net architecture directly. The i960 maintains a data structure holding the protection information for all open endpoints. 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 I/O bus, 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 control interface to U-Net on the i960 consists of a single i960-resident command queue that is only accessible from the kernel. Processes use the system call interface to the device driver that implements the kernel resident part of U-Net. This driver assists in providing protection by validating requests for the creation of communication segments and related endpoints, and by providing a secure interface between the operating system service that manages the multiplexing tags and the U-Net channel registration with the i960. The tags used for the ATM network consist of a VCI pair that implements full duplex communication (ATM is a connection oriented network and requires explicit connection set-up even though U-Net itself is not connection oriented). 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.

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 packet and the channel identifier to the i960. Single cell packet sends are optimized in the firmware because many small messages are less than a cell in size. For larger sized messages, the host-i960 DMA uses three 32-byte burst transfers to fetch two cells at a time and computes the AAL5 CRC using special SBA-200 hardware.

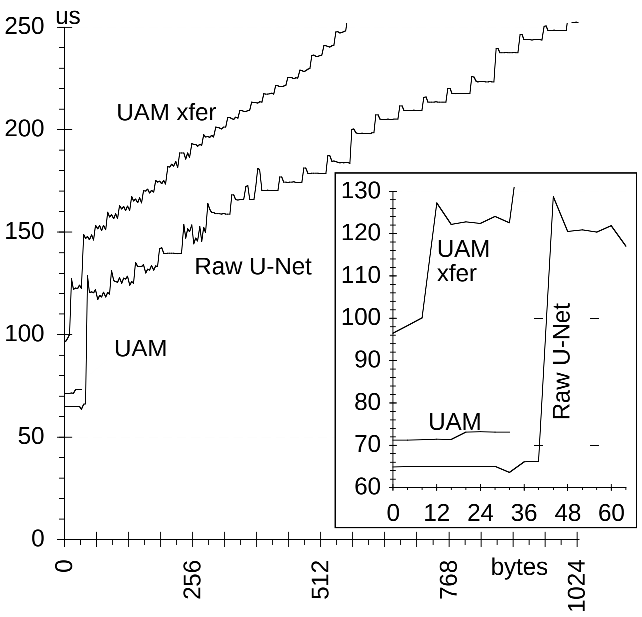

Figure 3: U-Net round-trip times as a function of message size. The *Raw U-Net* graph shows the round-trip times for a simple ping-pong benchmark using the U-Net interface directly. The inset graph highlights the performance on small messages. The *UAM* line measures the performance of U-Net Active Messages using reliable single-cell requests and replies whereas *UAM xfer* uses reliable block transfers of arbitrary size.

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 the entire message—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 packet is received, the message descriptor containing the pointers to the buffers is DMA-ed into the next receive queue entry.

4.2.3 Performance

Figure 3 shows the round trip times for messages up to 1K bytes, i.e., the time for a message to go from one host to another via the switch and back. The round-trip time is 65µs 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 placed on the memory bus (Mbus) allow round-trips in 12µs. Using a UNIX signal to indicate message arrival instead of polling adds approximately another 30µs on each end. Longer messages start at 120µs for 48 bytes and cost roughly an extra 6µs per additional cell (i.e., 48 bytes). Figure 4 shows the bandwidth over the raw base level U-Net interface in Mbytes/sec for message sizes varying from 4 bytes to 5Kbytes. It is clear from the graph that with packet sizes as low as 800 bytes, the fiber can be saturated.

4.2.4 Memory requirements

The current implementation pins pages used in communication segments down to physical memory and maps them into the SBA-200's DMA space. In addition, each endpoint has its own set of

[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.

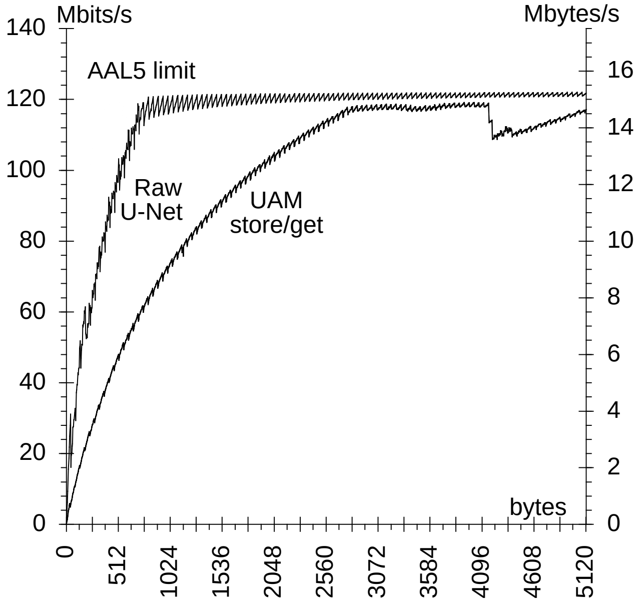

Figure 4: U-Net bandwidth as a function of message size. The *AAL-5 limit* curve represents the theoretical peak bandwidth of the fiber (the sawtooths are caused by the quantization into 48-byte cells). The *Raw U-Net* measurement shows the bandwidth achievable using the U-Net interface directly, while *UAM store/get* demonstrate the performance of reliable U-Net Active Messages block transfers.

send, receive and free buffer queues, two of which 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 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.

5 U-Net Active Messages implementation and performance

The U-Net Active Messages (UAM) layer is a prototype that conforms to the Generic Active Messages (GAM) 1.1 specification[9]. 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 live-lock, 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 block gets and stores. GAM provides reliable 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.

5.1 Active Messages implementation

The UAM 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 mainly performs the flow-control and retransmissions necessary to implement reliable delivery and the Active Messages-specific handler dispatch.

5.1.1 Flow Control Issues

In order to ensure reliable message delivery, UAM uses a window-based flow control protocol with a fixed window size (*w*). Every endpoint preallocates a total of 4*w* transmit and receive buffers for every endpoint it communicates with. This storage allows *w* requests and *w* replies to be kept in case retransmission is needed and it allows 2*w* request and reply messages to arrive without buffer overflow.

Request messages which do not generate a reply are explicitly acknowledged and a standard “go back N” retransmission mechanism is used to deal with lost requests or replies. 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.

5.1.2 Sending and Receiving

To send a request message, UAM 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 the send window is full, the sender polls for incoming messages until there is space in the send window or until a time-out occurs and all unacknowledged messages are retransmitted. The sending of reply messages or explicit acknowledgments is similar except that no flow-control window check is necessary.

The UAM layer receives messages by explicit polling. On message arrival, UAM 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.

5.2 Active Messages micro-benchmarks

Four different micro-benchmarks were run to determine the round trip times and block transfer bandwidths:

For single cell messages, the round trip time for bulk transfers, the bandwidth for bulk store and the bandwidth for bulk get operations.

1. The single-cell round trip time was estimated by repeatedly sending a single cell request message with 0 to 32 bytes of data to a remote host specifying a handler that replies with an identical message. The measured round trip times are shown in Figure 3 and start at 71µs which suggests that the UAM overhead over raw U-Net is about 6µs. This includes the costs to send a request message, receive it, reply and receive the reply.
2. The block transfer round-trip time was measured similarly by sending messages of varying sizes back and forth between two hosts. Figure 3 shows that the time for an *N*-byte transfer is roughly $135\mu s + N \cdot 0.2\mu s$. The per-byte cost is higher than for Raw U-Net because each one-way UAM transfer involves two copies (from the source data structure into a send buffer and from the receive buffer into the destination data structure).
3. The block 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. Figure 4 shows that the bandwidth reaches 80% of the AAL-5 limit with blocks of about 2Kbytes.

The dip in performance at 4164 bytes is caused by the fact that UAM uses buffers holding 4160 bytes of data and thus additional processing time is required. The peak bandwidth at 4Kbytes is 14.8Mbytes/s.

4. The block get bandwidth was measured by sending a series of requests to a remote node to fetch a block of specified size and waiting until all blocks arrive. The block get performance is nearly identical to that of block stores.

5.3 Summary

The performance of Active Messages shows that the U-Net interface is well suited for building higher-level communication paradigms used by parallel languages and run-times. The main performance penalty of UAM over raw U-Net is due to the cost of implementing reliability and removing the restrictions of the communication segment size: UAM must send acknowledgment messages and it copies data into and out of buffers in the communication segment. For large transfers there is virtually no bandwidth loss due to the extra copies, but for small messages the extra overhead of the copies and the acknowledgments is noticeable.

Overall, the performance of UAM is so close to raw U-Net that using the raw interface is only worthwhile if control over every byte in the AAL-5 packets is required (e.g., for compatibility) or if significant benefits can be achieved by using customized retransmission protocols.

6 Split-C application benchmarks

Split-C[7] is a simple parallel extension to C for programming distributed memory machines using a global address space abstraction. It is implemented on top of U-Net Active Messages and is used here to demonstrate the impact of U-Net on applications written in a parallel language. A Split-C program comprises one 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, IBM SP-2, and Cray T3D supercomputers as well as over U-Net Active Messages. A small set of application benchmarks is used here to compare the U-Net version of Split-C to the CM-5[7,29] and Meiko CS-2[25] 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 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.

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	71µs	14Mb/s

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

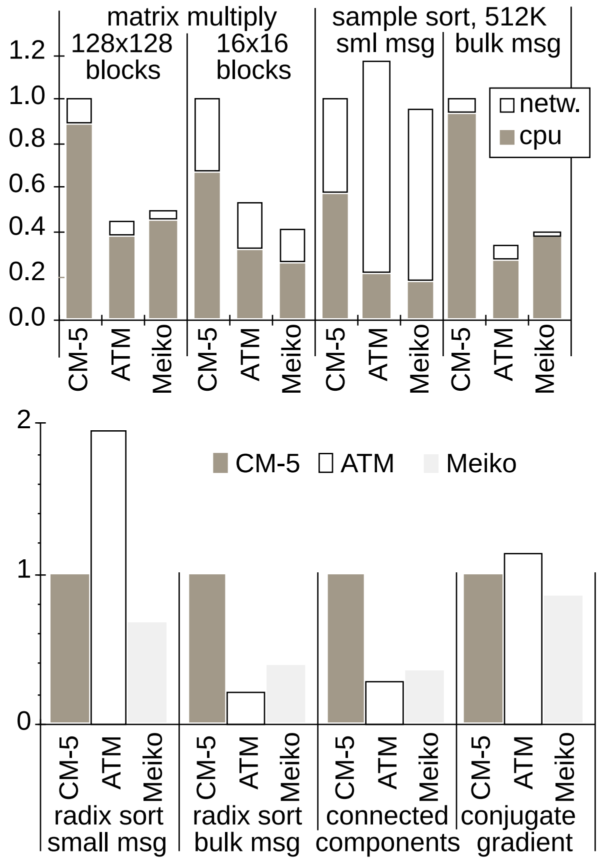

Figure 5: Comparison of seven Split-C benchmarks on the CM-5, the U-Net ATM cluster, and the Meiko CS-2. The execution times are normalized to the CM-5 and the computation/communication breakdown is shown for three applications.

The Split-C benchmark set used here is comprised of seven programs: a blocked matrix multiply[7], a sample sort optimized for small messages[8], the same sort optimized to use bulk transfers[25], two radix sorts similarly optimized for small and bulk transfers, a connected components algorithm[20], 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 execution times for runs on eight processors are shown in Figure 5; 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 samples the keys, then permutes all keys, and finally sorts the local keys on each processor. The version optimized for small messages packs two values per message during the permutation phase while the one optimized for bulk transfers presorts the local values such that each processor sends exactly one message to every other processor. 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.

7 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 the IP protocol suite plays a central role, its availability on many platforms provides a portable base for large classes of applications. Benchmarks are available to test the various TCP/IP and UDP/IP implementations, with a focus on bandwidth and latency as a function of application message size.

Unfortunately the performance of kernelized UDP and TCP protocols in SunOS combined with the vendor supplied ATM driver software has been disappointing: the maximum bandwidth with UDP is only achieved by using very large transfer sizes (larger than 8Kbytes), while TCP will not offer more than 55% of the maximum achievable bandwidth. The observed round-trip latency, however, is even worse: for small messages the latency of both UDP and TCP messages is larger using ATM than going over Ethernet: it simply does not reflect the increased network performance. Figure 6 shows the latency of the Fore-ATM based protocols compared to those over Ethernet.

TCP and UDP modules have been implemented for U-Net using the base-level U-Net functionality. The low overhead in U-Net protocol processing and the resulting low-latency form the basis for TCP and UDP performance that is close to the raw U-Net performance limits presented in §4.2.

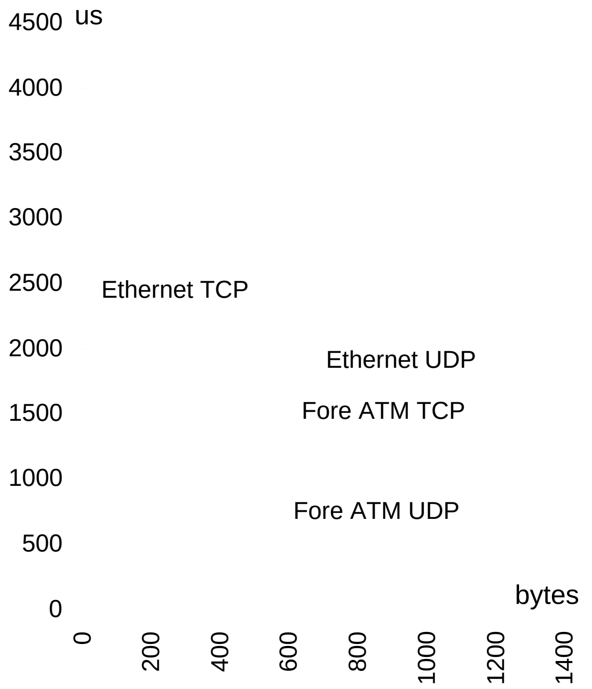

Figure 6: TCP and UDP round-trip latencies over ATM and Ethernet. as a function of message size.

7.1 A proof-of-concept implementation.

The TCP and UDP over U-Net implementation effort has two goals: first to show that the architecture is able to support the implementation of traditional protocols and second to create a test environment in which traditional benchmarks can be used to put U-Net and kernelized protocol processing into perspective.

By basing the U-Net TCP & UDP implementation on existing software [5] full protocol functionality and interoperability is maintained. A number of modules that were not in the critical performance path were not ported to U-Net, namely the ARP and ICMP modules.

At this point the secure U-Net multiplexor does not have support for the sharing of a single VCI among multiple channels, making it impossible to implement the standard IP-over-ATM transport mechanism which requires a single VCI to carry all IP traffic for all applications [21]. For IP-over-U-Net a single channel is used to carry all IP traffic between two applications, which matches the standard processing as closely as currently possible. This test setup does not use an exclusive U-Net channel per TCP connection, although that would be simple to implement.

Work is in progress to resolve the IP-over-ATM incompatibility and to implement proper ICMP handling when targeting IPv6 over U-Net. For this an additional level of demultiplexing is foreseen and will be based on the IPv6 [flow-id, source address] tag when packets arrive over the dedicated IP-over-ATM VCI. Packets for which the tag does not resolve to a local U-Net destination will be transferred to the kernel communication endpoint for generalized processing and possibly triggering ICMP handling. This will yield an implementation that is fully interoperable with other IP-over-ATM implementations and will cover both local and wide-area communication.

7.2 The protocol execution environment.

The TCP/IP suite of protocols is frequently considered to be ill-suited for use over high-speed networks such as ATM. However, experience has shown that the core of the problems with TCP/IP performance usually lie in the particular *implementations* and their *integration* into the operating system and not with the protocols themselves. This is indeed the case with the Fore driver software which tries to deal with the generic low-performance buffer strategies of the BSD based kernel.

Using U-Net the protocol developer does not experience a restrictive environment (like the kernel) where the use of generalized buffer and timer mechanisms is mandatory and properties of network and application can not be incorporated in the protocol operation. U-Net gives the developer the freedom to design protocols and protocol support software such as timer and buffer mechanisms, that are optimized for the particular application and the network technology used. This yields a toolbox approach to protocol and application construction where designers can select from a variety of protocol implementations.

As a result, U-Net TCP and UDP deliver the low-latency and high bandwidth communication expected of ATM networks without resorting to excessive buffering schemes or the use of large network transfer units, while maintaining interoperability with non-U-Net implementations.

7.3 Message handling and staging.

One of the limiting factors in the performance of kernel based protocols is the bounded kernel resources available, which need to be shared between many potential network-active processes. By implementing protocols at user-level, efficient solutions are avail-

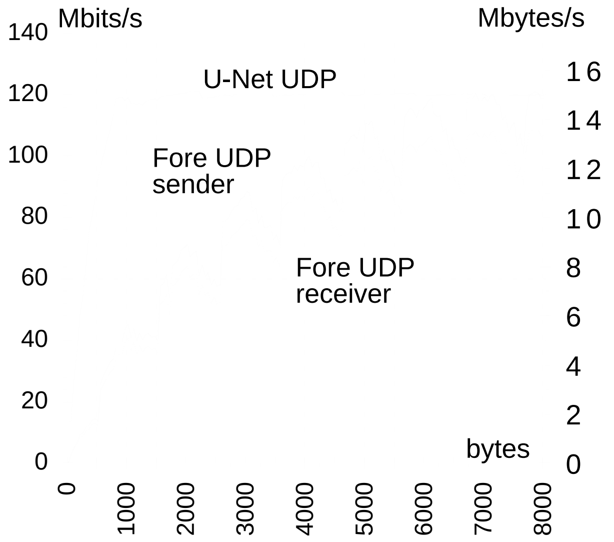

Figure 7: UDP bandwidth as a function of message size.

able for problems which find their origin in the use of the operating system kernel as the single protocol processing unit. Not only does U-Net remove all copy operations from the protocol path but it allows for the buffering and staging strategies to depend on the resources of the application instead of the scarce kernel network buffers.

An example is the restricted size of the socket receive buffer (max. 52Kbytes in SunOS), which 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 be tuned to meet application needs and be efficiently incorporated into the end-to-end flow-control mechanisms.

The deficiencies in the BSD kernel buffer (*mbuf*) mechanism have been identified long ago [11] and the use of high-performance networks seem to amplify the impact of this mechanism even more, especially in combination with the Fore driver buffering scheme. Figure 7 shows the UDP throughput with the saw-tooth behavior that is caused by the buffer allocation scheme where first large 1Kbyte buffers are filled with data and the remainder, if less than 512 bytes, is copied into small mbufs of 112 bytes each. This allocation method has a strong degrading effect on the performance of the protocols because the smaller mbufs do not have a reference count mechanism unlike the large cluster buffers.

Although an alternative kernel buffering mechanism would significantly improve the message handling in the kernel and certainly remove the saw-tooth behavior seen in Figure 7, it is questionable if it will contribute as significantly to latency reduction as, for example, removing kernel-application copies entirely [18].

Base-level U-Net provides a scatter-gather message mechanism 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 added by the TCP and UDP support software allows them to be shared by several messages without the need for copy operations.

7.4 Application controlled flow-control and feedback.

One the major advantages of integrating the communication subsystem into the application is that the application can be made aware of the state of the communication system and thus can take application specific actions to adapt itself to changing circumstances. Kernel based communication systems often have no other facility than to block or deny a service to an application, without being able to communicate any additional information.

At the sending side, for example, feedback can be provided to the application about the state of the transmission queues and it is simple to establish a back-pressure mechanism when these queues reach a high-water mark. Among other things, this overcomes problems with the current SunOS implementation which will drop random packets from the device transmit queue if there is overload without notifying the sending application.

Other protocol specific information such as retransmission counters, round trip timers, and buffer allocation statistics are all readily available to the application and can be used to adapt communication strategies to the status of the network. The receive window under U-Net/TCP, for example, is a direct reflection of the buffer space at the application and not at the intermediate processing unit, allowing for a close match between application level flow control and the receive-window updates.

7.5 IP

The main functionality of the IP protocol is to handle the communication path and to adapt messages to the specifics of the underlying network. On the receiving side IP-over-U-Net is liberal in the messages that it accepts, and it implements most of the IP functionality, except for the forwarding of messages and the interfacing to ICMP. A transport protocol is selected and the U-Net demultiplex information is passed on to the transport module to possibly assist in destination selection.

On the sending side the functionality of the IP protocol is reduced to mapping messages into U-Net communication channels. Because of this reduced functionality, this side of the protocol is collapsed into the transport protocols for efficient processing.

IP over U-Net exports an MTU of 9Kbytes and does not support fragmentation on the sending side as this is known to be a potential source for wasting bandwidth and triggering packet retransmissions [19]. TCP provides its own fragmentation mechanism and because of the tight coupling of application and protocol module it is relatively simple for the application to assist UDP in achieving the same functionality.

7.6 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 endpoint information passed-on by U-Net. A simple pcb caching scheme per incoming channel allows for significant processing speedups, as described by [23]. The checksum adds a processing overhead of 1µs per 100 bytes on a SPARCStation 20 and can be combined with the copy operation that retrieves the data from the communication segment. It can also be switched off by applications that use data protection at a higher level or are satisfied with the 32-bit CRC at the U-Net AAL5 level.

The performance of U-Net UDP is compared to the kernel based UDP in Figures 6 and 7. 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 and receiving hosts. With the same experimental set-up, U-Net UDP does not experience any losses and only the receive bandwidth is shown.

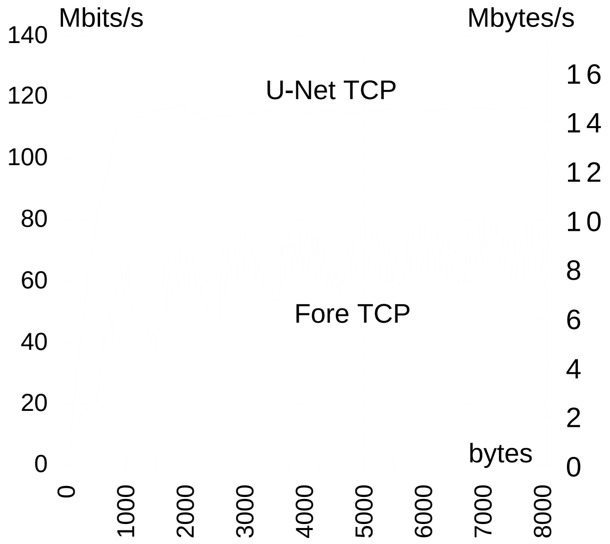

Figure 8: TCP bandwidth as a function of data generation by the application.

7.7 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.

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 8 shows that in most cases U-Net TCP achieves a 14-15 Mbytes/sec bandwidth using an 8Kbyte window, while even with a 64K window the kernel TCP/ATM combination will not achieve more than 9-10 Mbytes/sec. The round-trip latency performance of both kernel and U-Net TCP implementations is shown in Figure 9 and highlights the fast U-Net TCP round-trip which permits the use of a small window.

7.8 TCP tuning.

TCP over high-speed networks has been studied extensively, especially over wide-area networks [17] 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. These changes need to be incorporated into the U-Net TCP implementation if it is to function across wide-area links where the high latencies no longer permit the use of small windows.

It has been argued lately that the same changes are also needed for the local area case in order to address the deficiencies that occur because of the high latency of the ATM kernel software. U-Net TCP shows that acceptable performance can be achieved in LAN and MAN settings without any modifications to the general algorithms, without the use of large sequence numbers, and without extensive buffer reservations.

Tuning a number of the TCP transmission control variables is not without risk when running over ATM [24] and should be done with extreme caution. The low latency of U-Net allows for very conservative settings, therefore minimizing the risk while still achieving maximum performance.

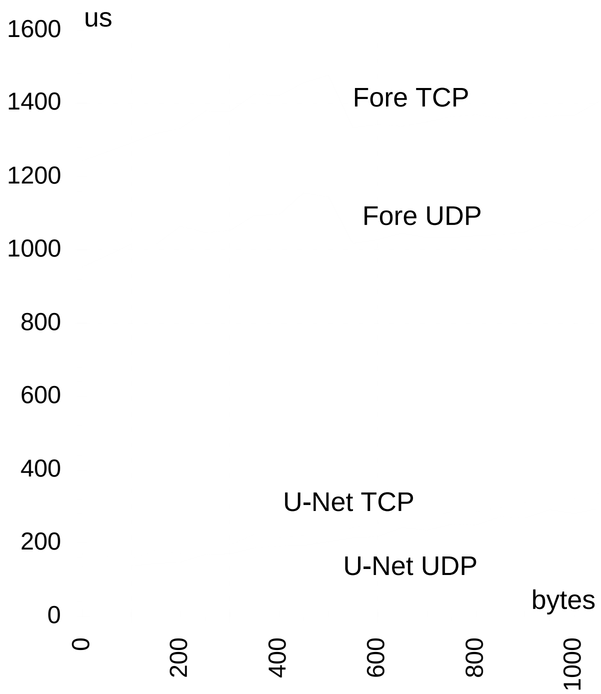

Figure 9: UDP and TCP round-trip latencies as a function of message size.

An important tuning factor is the size of the segments that are transmitted: using larger segments it is more likely that the maximum bandwidth can be achieved in cases where low latency is not available. Romanov & Floyd's work however has shown that TCP can perform poorly over ATM if the segment size is large, 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 the bandwidth shown in Figure 8.

Another popular approach to compensate for high latencies is to grow the window size. This allows a large amount of data to be outstanding before acknowledgments are expected back in the hope to keep the communication pipe filled. Unfortunately, increasing the window has a number of drawbacks. First of all, the large amount of data must be buffered to be available for retransmission. Furthermore, there is a risk of triggering the standard TCP congestion control mechanism whenever there are two or more segments dropped within a single window. Tuning the window size to a large value will increase the chance of this situation occurring, resulting in a drain of the communication pipe and a subsequent slow-start. It seems unavoidable to run these risks, even in a LAN setting, when the protocol execution environment is not able to guarantee low-latency communication.

A final tuning issue that needed to be addressed 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 BSD 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 is 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

Protocol	Round-trip latency	Bandwidth 4K packets
Raw AAL5	65µs	120Mbits/s
Active Msgs	71µs	118Mbits/s
UDP	138µs	120Mbits/s
TCP	157µs	115Mbits/s
Split-C store	72µs	118Mbits/s

Table 3: U-Net latency and bandwidth Summary.

timer granularity which is constrained by the reliability of the user-level timing mechanisms.

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. In U-Net TCP it was possible to disable the delay mechanism and thereby achieve more reliable performance. Disabling this bandwidth conserving strategy is justified by the low cost of an active acknowledgment, which consists of only a 40 byte TCP/IP header and thus can be handled efficiently by single-cell 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. In wide-area settings, however, the bandwidth conservation may play a more important role and thus the delayed acknowledgment scheme may have to be enabled for those situations.

8 Summary

The two main objectives of U-Net—to provide efficient low-latency communication and to offer a high degree of flexibility—have 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. Table 3 summarizes the various U-Net latency and bandwidth measurements. U-Net presents a simple network interface architecture which simultaneously supports traditional inter-networking protocols as well as novel communication abstractions like Active Messages.

Using U-Net the round-trip latency for messages smaller than 40 bytes is about 65 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 800 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 new flexibility in addition to high performance. 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.

9 Acknowledgments

U-Net would not have materialized 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 Infrastructure Grant CDA-9216202. The CM-5 results were obtained on the UCB machine, funded under NSF Infrastructure Grant CDA-8722788. 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. Werner Vogels is supported under ONR contract N00014-92-J-1866. We also thank Fore Systems for making the source of the SBA-200 firmware available to us.

In the final phase several reviewers and Deborah Estrin, our SOSP shepherd, provided helpful comments, in particular on TCP/IP.

10 References

[1] M. Abbot and L. Peterson. *Increasing Network Throughput by Integrating Protocol Layers.* IEEE/ACM Transactions on Networking. Vol. 1, No. 5, pages 600-610, Oct. 1993.

[2] 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.

[3] M. Bailey, B. Gopal, M. Pagels, L. Peterson and P. Sarkar. *PATHFINDER: A pattern Based Packet Classifier.* Usenix Symposium on Operating Systems Design and implementation, pages 115-124, Nov. 1994.

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

[5] L. Brakmo, S. O'Malley and L. Peterson. *TCP Vegas: New Techniques fro Congestion Detection and Avoidence.* In Proc. of SIGCOMM-94, pages 24-35, Aug 1994.

[6] T. Braun and C. Diot. *Protocol Implementation Using Integrated Layer Processing.* In Proc. of SIGCOMM-95, Sept 1995.

[7] 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.

[8] 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.

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

[10] D. Clark and D. Tennenhouse. *Architectural Considerations for a New Generation of protocols*. In Proc. of SICOMM-87, pages 353-359, Aug. 1987.

[11] D. Clark, V. Jacobson, J. Romkey, H. Salwen. *An Analysis of TCP Processing Overhead.* IEEE Communications, pages 23-29, June 1989.

[12] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards and J. Lumley. *Afterburner.* IEEE Network Magazine, Vol 7, No. 4, pages 36-43, July 1993.

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

[14] P. Druschel, 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.

[15] 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

[16] S. Floyd, V. Jacobson and S. McCanne. *A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing.* In Proc. of SIGCOMM-95, Sept 1995.

[17] V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*. IETF Request for Comments 1323, May 1992.

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

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

[20] 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

[21] M. Laubach. *Classical IP and ARP over ATM*. IETF Request for Comments 1577, January 1994.

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

[23] C. Partridge and S. Pink. *A Faster UDP.* IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pages 429-440, Aug. 1993.

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

[25] K. E. Schauser and C. J. Scheiman. *Experience with Active Messages on the Meiko CS-2.* In Proc. of the 9th International Parallel Processing Symposium (IPPS'95), Santa Barbara, CA, April 1995.

[26] Brian C. Smith. *Cyclic-UDP: A Priority-Driven Best-Effort Protocol.* http://www.cs.cornell.edu/Info/Faculty/bsmith/nossdav.ps.

[27] 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.

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

[29] 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.