Incorporating Memory Management into User-Level Network Interfaces

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

User-level network interfaces allow applications direct access to the network without operating system intervention on every send and receive. Messages are transferred directly to and from user-space by the network interface while observing the traditional protection boundaries between processes. First generation user-level network interfaces limit this message transfer to a per-process region of permanently-pinned physical memory to allow safe DMA. This approach is inflexible in that it requires data to be copied into and out of this memory region, and does not scale to a large number of processes.

This paper presents an extension to the U-Net user-level network architecture (U-Net/MM) allowing messages to be transferred directly to and from any part of an application's address space. This is achieved by integrating a translation look-aside buffer into the network interface and coordinating its operation with the operating system's virtual memory subsystem. This mechanism allows network buffer pages to be pinned and unpinned dynamically. Two implementations of U-Net/MM are described, demonstrating that existing commodity hardware and commercial operating systems can efficiently support the architecture.

1 Introduction

Recent research in high-speed network interfaces for commodity networks has focused on removing the operating system from the critical path for sending and receiving messages. An effective solution is to provide user-level messaging [1,3,4,7,9,10,12,15,16,17,19] where user applications can directly access the network hardware without violating protection. This allows messages to be sent from and received to user-space without kernel intervention. One of the main implementation difficulties of user-level network interfaces (NIs) is managing the mapping between the virtual addresses of message buffers specified by applications and the physical addresses required for actual transmission and reception. This includes two major components: the NI must be able to translate the virtual buffer addresses to physical addresses and the translations must be coordinated with the operating system's virtual memory subsystem.

So far, two solutions have been proposed which more or less side-step the issue. Custom designs appropriate for "next generation" systems have integrated the NI into the virtual-address side of the system where it shares or replicates the CPU's TLB [1,12,16]. Less aggressive proposals allocate a physically-contiguous buffer region for each application and pin it down to physical memory [3,4,13,17]. The application specifies message buffers using offsets into the buffer region which the network interface can easily bounds-check and translate. This prevents the NI from issuing illegal DMA accesses because the virtual to physical mapping is fixed and all buffer pages are guaranteed to have physical backing.

The work presented here addresses the memory management issues on commodity processors and networks where the network interfaces reside on the physical-address side of the system (typically the I/O bus) and are independent of the processor architecture. In these systems the NI transfers data to and from message buffers in main memory using DMA which requires physical memory addresses. If applications are to interact with the NI directly, the latter must incorporate some form of address translation. Moreover, for portability reasons, it is desirable that the address translation be independent of the host processor architecture, especially the host's page table representation. This precludes the type of solutions used in parallel machines with custom networks and NIs on the system bus or even integrated on the CPU chip.

The major contribution of this paper is two-fold:

- It describes a user-level NI design based on the U-Net architecture [17], called U-Net/MM, which incorporates a TLB into the NI to overcome the limitations of pinned-down communication buffers. Applications can use arbitrary virtual memory addresses for message sends and receives and only the pages covered by the NI TLB are

For additional information please refer to `http://www2.cs.cornell.edu/U-Net/`

locked down. The total memory reserved for the network is primarily a function of the TLB size and not of the number of processes running.

- It compares the performance of two complete implementations of the design, the first using custom firmware for an i960 co-processor in the NI and the second using a simpler NI and fast traps into the kernel. The implementations span the design space and offer an understanding of the issues involved in integrating a TLB into the NI.

The remainder of this paper is organized as follows. The motivation for incorporating memory management into the network interface is presented in Section 2 and related work is discussed in Section 3. Section 4 introduces the new U-Net/MM network interface architecture and Section 5 describes the two implementations which are then benchmarked in Section 6. Section 7 discusses the design space and concludes the paper.

2 Motivation

Kernel-based networking architectures have centralized control of the network buffers that are shared by all applications. The buffer pool is roughly proportional in size to the bandwidth of the network and is typically preallocated and pinned to physical memory so that the network interface can asynchronously transfer data to and from these buffers. In the case of user-level networking, however, the network buffers are part of the user address space and can no longer be shared. Most user-level network implementations continue to pin the pages used for network buffers which implies that the total amount of memory dedicated for communication grows not only with the network bandwidth but also with the number of applications accessing the network.[1] While only very few applications can simultaneously make significant use of the network, a typical UNIX workstation has dozens of processes with open network connections, most of which are sleeping at any given point in time.

The benefits of paging network buffers are twofold: (i) it increases scalability in the number of concurrent network applications by allowing buffers belonging to inactive applications to be paged out, and (ii) it enables *zero-copy* sends by allowing applications to send from arbitrary virtual addresses as opposed to current implementations that require an extra copy of data from the application data structures to the pinned network buffer area. The receive model for U-Net/MM allows zero-copy receives from the network into a pool of application specified receive buffers. However, the application cannot control which specific buffer in the pool a particular message is received into. Thus, true zero-copy receives directly into application data structures may not be always possible. While conceptually simple, paging network buffers raises two major issues: (i) since the NI accesses network buffers asynchronously, the virtual memory mapping operations performed by the kernel need to be coordinated with the NI operation, and (ii) the effect of paging network buffers on communication overheads needs to be examined.

From an architecture point of view, this work explores the design space of network interfaces. Previous-generation commodity network interfaces were simple DMA engines on the I/O bus. With the increasing complexity of high speed networks, network interfaces are becoming more sophisticated and routinely include an on-board co-processor, although such co-processors usually lag behind the host CPU in performance. This raises the question of how much functionality should be implemented in the NI itself and how much should be relegated to the faster host. Shared-memory multiprocessors suggest an alternate design point where one of the processors can be dedicated to managing a “dumb” NI and performing some protocol processing [11][16]. This paper proposes an intermediate model which transfers just enough intelligence into the NI to do message mux/demux and virtual address translation, thereby enabling scalable, zero-copy messaging.

3 Related Work

Various techniques have been used to address memory-management issues in the context of user-level networking. They range from closely coupling the NI with the processor TLB to incorporating a TLB in the NI. Other approaches have used operating system assisted dynamic remapping of network buffers to the address space of user processes.

In the StarT [1] project, the NI is integrated onto the main processor chip and shares its TLB directly. The NI in FLASH [12] does not include a TLB but can validate physical addresses provided by an application. In the Typhoon [16] architecture the network processor includes a full memory management unit, but the consistency issues are sidestepped by swapping complete buffer areas instead of paging. The Meiko CS-2 [2,10,14] NI is close to the design proposed in this paper: the NI includes a TLB for virtual to physical address translation and TLB misses are handled

1. In a multiprogrammed environment it is not unreasonable for applications to allocate enough buffers to prevent buffer overrun while another process is running. At now-common network rates of over 100Mbit/s, a megabyte of receive buffers can be filled in under 100ms which corresponds to only a few time-slices.

by the operating system (a customized version of Solaris). Unfortunately no implementation details or performance data on the TLB performance or the kernel modifications are available in the literature.

The important features of the U-Net/MM architecture are the use of a TLB in the NI, its independence of the page table format, and the handling of the TLB consistency issues. In addition, all the above architectures involve custom hardware implementations for high-performance networks in parallel machines while U-Net adapts off-the-shelf network interfaces and commodity networks.

An alternative approach [3,8] uses commodity network interfaces and specialized operating system support to dynamically remap network buffers to address spaces of user-level processes. A major optimization in the *fbufs* [8] approach assumes that pages from only a limited range of user virtual memory can be remapped while in the case of SHRIMP [3], any region in the user virtual memory can be used as a network buffer but must be explicitly pinned down in physical memory using special system calls. In contrast, the U-Net/MM architecture allows applications to send from and receive into any area in their virtual address space without having to explicitly pin down any virtual memory region.

4 The U-Net/MM Architecture

U-Net/MM is an extension of the U-Net user-level networking architecture [17]. It consists of three main building blocks shown in Figure 1: *endpoints* serve as an application's handle into the network and contain three *message queues* which hold descriptors for *message buffers* that are to be sent, that are free for reception, and that have been received. Each process that wishes to access the network first creates one or more endpoints, each with an associated set of message queues. Communication to and from an endpoint uses *message tags* to uniquely identify network sources and destinations. The NI uses these tags to verify the destination address of outgoing messages as well as demultiplex incoming messages to the appropriate endpoint. The exact form of a message tag depends on the network substrate — for example, for ATM networks, virtual channel identifiers (VCIs) may be used. An application registers the message tags with U-Net before sending or receiving messages— an operating system service is needed to assist the application in determining the correct tag to use based on the destination process and the route between the two communicating nodes.

In order to send, a process composes a message in one or more transmit buffers in its address space and pushes a descriptor onto the send queue. The descriptor contains pointers to the transmit buffers, their lengths and a destination tag. The network interface picks up the descriptor, translates the destination tag into a network header and the virtual buffer addresses to physical DMA addresses. It then transfers the data directly from the user-space buffer into the network.

When the NI receives data it examines the message header and matches it with the message tags to determine the correct destination endpoint. The NI then pops free buffer descriptors off the appropriate free queue, translates the virtual addresses, transfers the data into the buffers, and enqueues a descriptor onto the right receive queue. Applications can detect the arrival of messages by polling the receive queue, by blocking until a message arrives (e.g., a UNIX *select* system call), or by receiving an asynchronous notification on message arrival (e.g., a signal).

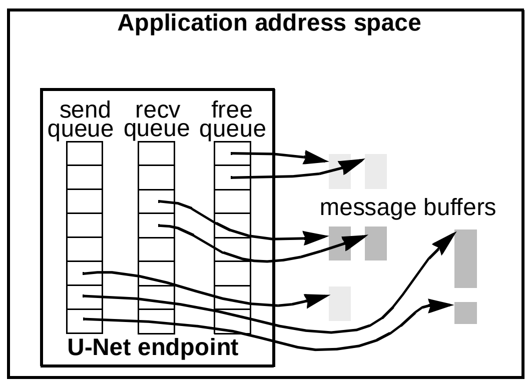


Figure 1: U-Net building blocks. *Endpoints* serve as an application's handle into the network, and message queues *(send/recv/free queues)* hold descriptors for messages that are to be sent or that have been received.

Unlike U-Net/MM, the original U-Net architecture does not allow use of arbitrary message buffers. Each endpoint is associated with a buffer area that is pinned to contiguous physical memory and holds all buffers used with that endpoint. Message descriptors contain offsets into the buffer area (instead of full virtual addresses) which are bounds-checked and added to the physical base address of the buffer area by the NI.

4.1 Address translation

In order to handle arbitrary user-space virtual addresses, the U-Net/MM design incorporates a Translation Lookaside Buffer (which is not directly visible from user-space) and mechanisms to handle TLB misses and TLB coherency issues. The U-Net/MM design itself is independent of any specific hardware platform.

The TLB is maintained by the network interface itself and maps <process ID, virtual address> pairs to physical page frames and read/write access rights. Whenever a physical page frame has an entry in the NI TLB, it is considered to be *mapped* into the corresponding endpoint, and the NI assumes that it is available for DMA operations.

During a transmit operation, the NI attempts to translate the virtual address of the send-buffer through a TLB lookup. In the case of a TLB hit, the transmission proceeds using the resulting physical address for DMA. In the case of a TLB miss, the NI requests a translation from the host kernel. If the page is in memory, the kernel returns the corresponding physical address and the NI completes the transmission. In the case of a privilege violation the kernel informs the NI, which propagates the error to the send completion status in the U-Net transmit queue. If the page is not memory-resident, the kernel initiates a page-in and notifies the NI to defer the transmission. The NI then suspends the processing of the affected endpoint until the page fault is resolved and the physical address of the buffer is supplied by the kernel.

In the case of message receives, the NI pre-translates a number of entries in each free buffer queue to minimize latencies. When a message arrives, the NI pops a pre-translated buffer address from the appropriate free queue and initiates a DMA transfer into main memory. If no pre-translated buffer is available, the message may be dropped. On receiving a translation request for a receive buffer that has been paged out, the kernel initiates a page-in and notifies the NI. If the page-in is not complete by the time the buffer is needed for message reception, the NI may skip the affected buffer and return it to the application unfilled to avoid stalling receive processing for the endpoint in question.

4.2 TLB Consistency

The overriding concern in the design of U-Net/MM was to find the right model for maintaining consistency between the NI TLB and the virtual memory system. Since it is difficult to change the virtual memory system of commercial operating systems, the U-Net/MM architecture uses existing operating system structures and avoids adding new functionality.

The idea underlying the consistency mechanism in U-Net/MM is to view the NI as another process which shares the set of network buffer pages used by applications. Pages for which the NI TLB has a valid mapping are pinned down by incrementing reference counts in the appropriate page table entries. The host kernel can impose an upper bound on the amount of memory pinned down in this fashion by limiting the number of valid entries in the NI TLB: this is somewhat analogous to preallocating and pinning a fixed-size buffer pool in kernel-based network stacks, the difference being that in the U-Net/MM case the set of pages in the buffer pool can vary with time.

On receiving a translation request from the NI, the kernel walks the page tables to establish the translation, the page is pinned (assuming it is present), and the physical address is returned to the NI. The kernel performs a copy-on-write if the NI requests a write-translation for a page that is read-only. Translation requests for non-present pages cause a page-in to be initiated. When the NI evicts a page from the TLB, it notifies the kernel, which decrements the reference count in the page descriptor and makes it available for swapping once more. The NI avoids evicting a page for which message transmission or reception is in progress in order to prevent illegal DMA accesses. The kernel never requests a mapping to be invalidated in the NI TLB (when an endpoint is closed, the NI invalidates the TLB entries held by that endpoint) thus ensuring atomicity of TLB invalidations with respect to message sends and receives.

An alternate approach would have been to treat the NI as another processor on the bus and use the same TLB coherency mechanisms used in shared memory multiprocessor (SMP) operating systems, in which the host kernel informs all processors of TLB invalidations when pages are unmapped. However, this mechanism is not well suited to the task as the NI performs page translations in a manner which is very different from the way a processor uses its TLB. In the SMP case, the TLB translates addresses on every memory access and invalidating a mapping in the middle of an instruction sequence which is using the mapping only has a minor performance impact. In contrast, the NI uses its TLB to set-up a DMA transfer which can last as long as it takes to transmit or receive a message. This implies that DMA transfers must be atomic relative to TLB invalidations (just as instructions are) which is infeasible because message reception time may not be bounded[1]. Dividing a DMA transfer into bursts, making each burst atomic, and allowing TLB invalidations in between bursts is not advisable either because a TLB miss in the middle of a message being received is likely to cause the message tail to be dropped and for outgoing messages the transmitter cannot necessarily be stalled without corrupting the packet. Finally, if the NI acts as a processor it must simulate reference bits used for page aging and no simple solution could be found for this problem.

1. For example, the cells of an ATM AAL5 PDU can arrive at an arbitrarily slow rate.

5 U-Net/MM Implementations

This section describes two implementations of the U-Net/MM architecture. The first uses a 155Mbps FORE Systems PCA-200 PCI ATM network interface which contains a programmable i960 processor and is hosted in a 133Mhz Pentium workstation running Linux v1.3.71. The second implementation uses a non-programmable Fast Ethernet interface with a DECchip 21140 controller and the same workstation running Windows NT 4.0 as host. Figure 2 depicts block diagrams of the two implementations which are described in detail in the next subsections.

5.1 PCA200/Linux implementation

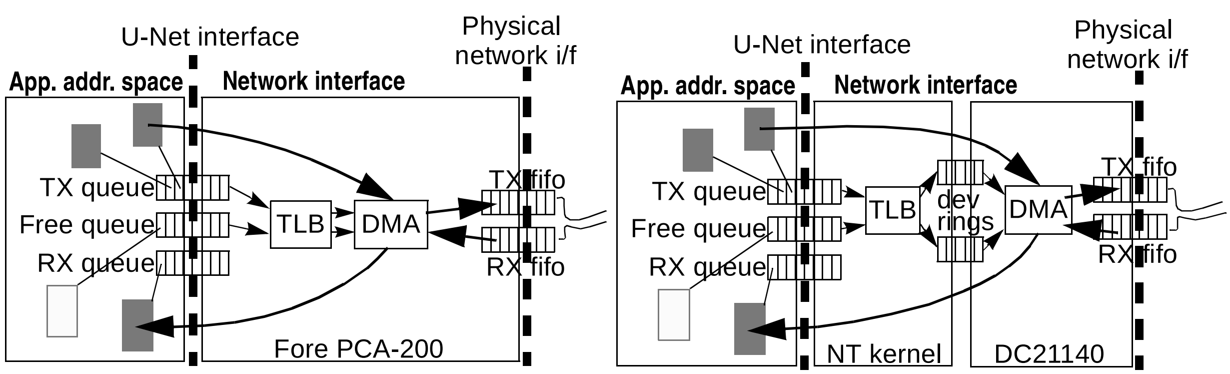
The PCA-200 contains a 25 MHz i960 processor, 256 Kbytes of RAM, a DMA-capable PCI bus interface, a simple FIFO interface to the 155Mbps (OC3) ATM fiber and an AAL5 CRC generator. U-Net/MM is implemented directly in the firmware of the i960 and in the Linux kernel. The i960 holds a data structure for each open endpoint pointing to the send and free queues allocated in i960 memory and the receive queue allocated in host memory. All queues are mapped into applications' address spaces which communicate with the i960 by directly writing to or reading from the queues.

The i960 firmware implements a two-level TLB consisting of a 1024-entry direct mapped primary table and a fully associative 16-entry secondary "victim cache". The latter uses a FIFO replacement policy and a mapping evicted from it is returned to the kernel for unpinning. Pages are pinned using a reference count which the Linux kernel associates with each page frame to indicate the number of processes sharing it. This reference count is only decremented if the page is removed from a process' virtual address space by the swapper task or through explicit unmapping. When it drops to zero, the page becomes a page-out candidate. U-Net/MM increments this reference count whenever the NI requests a translation for the page. The kernel does not maintain any page tables for the NI and thus the swapper cannot decrement the reference count of pages mapped by the NI TLB to zero. This ensures that they cannot be paged-out or otherwise unmapped until the NI removes the entry from its TLB and notifies the kernel which then decrements the reference count.

5.1.1 TLB miss handling

In case of TLB misses, the NI requests page translations from the kernel through a translation-request queue. If a victim-cache entry is to be evicted to make room for a new translation the request also indicates the page to unpin. The NI interrupts the kernel to service the queue and busy-waits until a mapping is returned.

To satisfy a page translation request from the NI, the kernel walks the page tables of the process owning the endpoint. If a present page exists, the kernel increments its reference count and returns the physical address to the NI immediately (in the case of read-access requests) or performs a copy-on-write operation (in the case of write-access requests to shared pages). In this way, a read or write translation by the NI is considered to be a read or write operation by the process owning the corresponding endpoint, allowing page-sharing and protection semantics to be preserved.



a) PCA-200/Linux implementation **b) DC21140/WindowsNT implementation**

Figure 2: Block diagrams of two U-Net/MM implementations.In the PCA-200/Linux case the application message queues are serviced directly by the NI which includes a TLB for address translation and a DMA engine. In the DC21140/WinNT case the address translation is performed in a kernel fast-trap. This fast trap controls the DMA engine in the NI through a pair of device buffer descriptor rings.

If the NI requests a translation for read access to a non-resident page the kernel immediately informs the NI that the translation will be deferred, and initiates the page-in from disk. Because page-ins cannot be initiated from within the interrupt handler, an in-kernel thread is signalled to handle the page-in and provide the eventual mapping to the NI. This thread is placed at the head of the run queue so it starts immediately after the interrupt handler returns. While the faulted translation is pending, the NI is free to poll other endpoints for transmission or to service incoming messages.

The current implementation makes the simplifying assumption that all receive buffers are page-aligned and page-sized. This allows the handling of page faults during receive-buffer translation to be handled by allocating a new zero page and mapping it into the process address space, instead of reading the missing page from disk.

5.2 DC21140/WinNT implementation

The Fast Ethernet network interface is based on the DECchip 21140 Fast Ethernet controller. The DC21140 is a PCI bus master capable of transferring complete frames to and from host memory via DMA and includes an Ethernet CRC generator/checker. The chip maintains a single set of circular send and receive rings containing descriptors pointing to buffers in host memory. The design of the DC21140 assumes that a single operating system agent will multiplex access to the hardware. Therefore, U-Net/MM must be largely implemented in the kernel where it multiplexes among multiple applications and translates from virtual to physical addresses.

The in-kernel implementation of U-Net is best thought of as sharing the physical CPU hardware between the regular host computation tasks and the new NI task. A fast kernel trap services the U-Net transmit queue in a manner similar to the i960 in the ATM implementation and an interrupt handler is triggered by the DC21140 when network packets arrive. The trap and interrupt handlers act as a protected co-routine that allows a portion of main processor time to be allocated to servicing U-Net.

The NI TLB is implemented in software in the kernel and has the same structure as that in the PCA-200/Linux implementation. In order to send a message, the application stores a descriptor into the U-Net send queue and issues a fast trap into the kernel U-Net service routine. This trap is implemented as an x86 trap gate and does not incur the overhead of a complete system call. The service routine traverses the queue and, for each entry, translates the buffer address and pushes corresponding descriptors onto the DC21140 send ring. After all descriptors have been pushed onto the device transmit ring, the service routine issues a transmit poll demand to the DC21140 which initiates the actual transmission. The service routine also monitors the U-Net free buffer queue and pre-translates addresses as needed to avoid a TLB miss at receive time.

Upon packet reception the DC21140 transfers the data into kernel buffers pointed to by the device's receive ring. These are a small number of fixed buffers allocated in the kernel and are used in FIFO order by the DC21140. The DC21140 generates an interrupt, the service routine determines the destination endpoint, fetches a free U-Net buffer, translates the address, copies the data into the buffer, and enqueues an entry in the user receive queue.

5.2.1 TLB miss handling

The kernel module services TLB misses using two functions exported by the Windows NT memory manager: *MmProbeAndLockPages* pins pages and *MmUnlockPages* unpins them. These functions are designed to support conventional I/O, which poses some challenges because *MmProbeAndLockPages* can only operate on pages of the currently-running process and cannot be executed within an interrupt context. This implies that a separate in-kernel thread cannot be used to monitor and prefetch free buffer pages; instead, this operation must be performed in the fast trap for the corresponding endpoint.

Entries evicted from the victim cache are unpinned by a call to *MmUnlockPages*, which can be invoked in the context of any running process. Evictions pose a problem with transmit buffers: once a buffer is enqueued on the transmit ring, its TLB entry cannot be evicted from the TLB victim cache until the transmit has completed without risking illegal DMA accesses. This situation is handled by implementing the victim cache as a linked-list which can grow to hold an arbitrary number of such pending entries. Whenever a TLB entry is pushed to the victim cache a cleanup routine is invoked to shrink the victim cache if it is over a certain size.

6 Performance

The performance measurements presented in this section serve two purposes: to determine whether the new address translation features in U-Net/MM add a measurable overhead to the critical path, and to establish the cost incurred in the case of TLB misses. In addition to the two U-Net/MM implementations described in the previous section, three conventional U-Net implementations without memory management in the NI were timed: a PCA-200 [20] and a

DC21140 [19][20] implementation using Linux as well as a DC21140 implementation using Windows NT. In all three implementations a permanently pinned buffer area is associated with each endpoint and the applications specify all buffer addresses as offsets relative to the buffer area.

6.1 Raw operation times

In order to determine the cost of individual operations in the U-Net implementations the code was instrumented using the Pentium cycle counters to time small sections of kernel code and an oscilloscope was placed on the PCI bus to observe DMA transfers[1]. The results are shown in Table 1. All measurements were taken on a 133Mhz Pentium system with a 33Mhz PCI bus.

			Overheads	Fore PCA-200 ATM Linux		DC21140 Linux	DC21140 Fast Ethernet Windows NT 4.0	
				U-Net	U-Net/MM	U-Net	U-Net	U-Net/MM
SEND			trap	—	—	1µs	3µs	3µs
			queue handling	10µs	10µs	3.2µs	2.8µs	2.8µs
			TLB hit	—	1µs	—	—	1µs
	TLB miss		NI-kernel handshake	—	24µs	—	—	—
			interrupt service	—	10µs	—	—	—
		pin page	present (pin)	—	11µs	—	—	16µs
			non-present (page fault)	—	~22ms	—	—	~22ms
			unpin page	—	4µs	—	—	4µs
PRE-TRANSLATE RECEIVE BUFFERS			TLB hit	—	1µs	—	—	—
	TLB miss		NI-kernel handshake	—	24µs	—	—	—
			interrupt service	—	10µs	—	—	—
		pin page	present non-shared (pin)	—	11µs	—	—	37µs
			present shared (copy-on-write)	—	44µs	—	—	
			non-present (zero map-in)	—	36µs	—	—	—
			non-present (page-fault)	—	—	—	—	~22ms
			unpin page	—	4µs	—	—	4µs
RECEIVE			interrupt entry	—	—	2µs	3µs	3µs
			queue handling	13µs	13µs	3.3µs	3.7µs	3.7µs
			TLB hit	—	2µs	—	—	1µs
			buffer copy	—	—	1.3µs-21µs	1.2µs-42µs	1.2µs-42µs

Table 1: Overheads of message transmission and reception with and without address translation in the NI.

In the case of U-Net/PCA-200 the only relevant timings are the times taken by the i960 to handle the transmit and receive queues as well as the associated DMA transfers. The U-Net/MM version on the same hardware and OS adds only 1-2µs in the case of a TLB hit. TLB misses, on the other hand cause on the order of 50µs to 100µs of delay (as long as no disk access is required): the translation request and reply handshake between NI and kernel, the interrupt overhead, and the time to pin and unpin a page account for almost 50µs. If the page contents must be copied or zeroed the overhead roughly doubles.

The implementations using the DC21140 do not incur any of the NI-kernel coordination overheads for memory management and the use of the faster host processor reduces the overheads considerably (although the main CPU is employed, not a separate co-processor.) The cost of the custom fast-traps into the kernel is surprisingly low, with Linux faster than NT due to the trap complexity. A further surprise is that receive buffer copy times (shown for 40 and 1498-byte messages) under Linux are half those of NT.

Overall the cost of memory management operations in Linux and NT are very comparable, although the Linux page-pinning routines have been custom-designed while under NT standard kernel routines were employed. One notable difference is the time to process a non-present write-page translation: under the PCA-200 implementation, receive buffers are assumed to be page-aligned, so the page fault can be avoided by simply mapping in a new zero

1. Measurements were made by observing the PCI FRAME# and REQ# signals on the PCA-200 and the DC21140, watching both regular and specially added “dummy” DMA accesses.

page. As multiple receive buffers may share a page in the DC21140 implementations (due to the MTU of Fast Ethernet) this optimization cannot be used.

6.2 Application Behavior

To obtain a preliminary characterization of the NI TLB behavior in the U-Net/MM architecture, a set of Split-C [5] benchmarks and the Linux kernel socket layer were instrumented to record message transmit and receive activity, the results of which were input to a simulation of the U-Net/MM TLB.

The Split-C language allows processes to access remote data by using *global pointers* — a virtual address coupled with a process identifier. Dereferencing a global pointer allows a process to read or write data in the address spaces of other processes that are part of the parallel application. Split-C is implemented over Active Messages [18], a low-overhead RPC mechanism which provides reliable communication.

The Split-C benchmark suite consists of four programs: a blocked matrix multiply, a radix sort and a sample sort optimized for small message transfers and the same sample sort optimized for large message transfers. The matrix multiplication was run twice, once using matrices of 8 by 8 blocks with 128 by 128 double floats in each block and once using 16 by 16 blocks with 16 by 16 double floats in each block. The radix and sample sort benchmarks sort an array of 32-bit keys over all nodes with 256K keys per node.

The Linux kernel sockets layer was instrumented to record buffer address and length for all stream and datagram socket send and receive operations, and traces captured for standard user interaction including *telnet*, *rlogin*, FTP and X-Window application sessions.

The TLB simulation results are shown on Figure 3 for 1024- and 256-entry direct-mapped primary TLBs each with a 16-entry fully-associative victim cache. 64 receive buffers, each of size 4Kb, are assumed. The number of non-compulsory misses for all the applications is zero for a 1024-entry primary TLB, implying that this is large enough to accommodate the mappings for all network buffer pages in a wide variety of applications. For the 256-entry TLB, some applications still show good TLB performance because they either send mostly small messages from a single page of 64-byte buffers (sample sort) or use a small number of pages as network buffers (only 82 pages are used by matrix multiply). The other applications show some non-compulsory read and write misses when the TLB size goes down to 256 entries. The Linux sockets trace includes a high number of receive compulsory misses which correspond to 64 receive buffer pages being consumed for each of the 32 processes recorded.

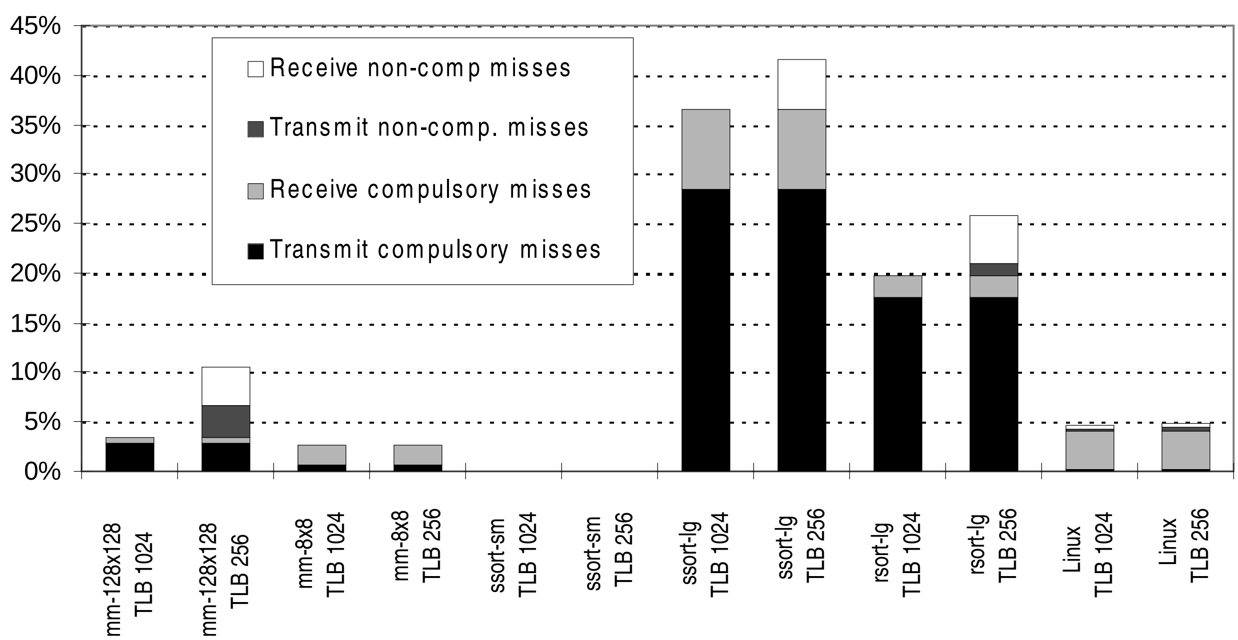


Figure 3: Simulated TLB performance for Split-C application benchmarks as well as for an interactive Linux workload.

7 Conclusion

U-Net/MM is an architecture for efficiently supporting scalable, protected user-level communication without the use of permanently-pinned memory segments. The architecture allows pages to be dynamically pinned and unpinned from physical memory as required by the network interface. A Translation Look-aside Buffer implemented in the NI handles mappings to these pages, and the OS kernel provides services to satisfy TLB translation and eviction requests by the NI. As an added benefit, user processes are able to directly transmit data out of any region of their address space, enabling true zero-copy sends. Two implementations of the U-Net/MM architecture, on network interfaces both with and without a programmable co-processor, have highlighted the performance trade-offs associated with the network interface design.

Experience with the two implementations of U-Net/MM highlights the trade-offs between reducing main CPU overhead through use of a network co-processor and reducing message latency with the memory-management system on the host. Using a programmable co-processor (as in the PCA-200) on the NI to securely multiplex the network hardware and do address translations reduces main CPU overhead but raises overall message latency because of communication over the I/O bus. Moreover, such co-processors often lag main processors in performance as shown by the 25MHz i960. On the other hand, using protected co-routines in the host kernel (as in the DC21140) reduces overall message latency at the cost of increasing main CPU overhead.

Although the U-Net/MM architecture supports zero-copy sends, it is infeasible to implement zero-copy receives into application data structures because of security reasons. The concern is to prevent corrupted or retransmitted messages from overwriting existing application data. U-Net/MM avoids this problem by receiving into user-designated receive buffers. In order to avoid an extra copy into application data structures, filled receive buffers can be used as long-lived data structures and new buffers may be allocated in other portions of the user address space.

8 Future work

The U-Net/MM architecture does not currently address the issue of process scheduling. In traditional kernel-based networking, the kernel is involved in every message receive and can coordinate process scheduling with network activity. For example, processes part of a parallel computation should be co-scheduled to improve overall application performance.

The U-Net/MM implementations described here represent two ends of the NI design spectrum: On one side, the PCA-200 performs both address translation and message multiplexing/demultiplexing in the network interface, while on the other, the DC21140 relies on the host for both functions. An interesting intermediate point would be a design which performs one of these operations directly on the NI, leaving the other to the host processor. The evaluation presented here suggests that it is desirable to perform message mux/demux on the NI directly to avoid buffer copies on receive, while using the host processor to perform address translation reduces communication latency and complexity.

Acknowledgments

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

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

References

[1] B. S. Ang, D. Chiou, L. Rudolph, and Arvind. Message Passing Support on StarT-Voyager. CSG-Memo-387, MIT Laboratory for Computer Science, July 1996.

[2] J. Beecroft, M. Homewood, and M. McLaren. Meiko CS-2 interconnect Elan-Elite design. *Parallel Computing,* 20(10-11): 1627-1638, 1994. Elsevier Science, Amsterdam, The Netherlands.

[3] M. Blumrich, C. Dubnicki, E. W. Felten, and K. Li. Virtual Memory Mapped Network Interfaces. *IEEE Micro,* 15(1):21-28, February 1995.

[4] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich and J. Wilkes. An Implementation of the Hamlyn Sender-Managed Interface Architecture. In *Proceedings of the 2nd Symposium on Operating System Design and Implementation,* Seattle, Washington, October, 1996.

[5] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Introduction to Split-C. In *Proceedings of Supercomputing '93,* 1993.

[6] D. E. Culler and G. M. Papadopoulos. The Explicit Token Store. *Journal of Parallel and Distributed Computing,* 10(4): 289-308, 1990.

[7] W. J. Dally, R. Davison, J. A. S. Fiske, G. Fyler, J. S. Keen, R. A. Lethin, M. Noakes, and P. R. Nuth. The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms. *IEEE Micro,* April 1990.

[8] P. Druschel and L. L. Peterson. Fbufs: A High-Bandwidth Cross-Domain Transfer Facility. In *Proceedings of the 14th Annual Symposium on Operating System Principles,* pages 189 - 202, Ashville, North Carolina, December 1993.

[9] M. Fillo, S. W. Keckler, W. J. Dally, N. P. Carter,A. Chang, Y. Gurevich, and W. S. Lee. The M-Machine Multicomputer. In *Proceedings of the 28th Annual International Symposium on Computer Micorarchitecture,* Ann Arbor, Michigan, 1995.

[10] M. Homewood and M. McLaren. Meiko CS-2 Interconnect Elan-Elite Design. In *Proceedings of Hot Interconnects,* August 1993.

[11] Intel Corporation. *Paragon XPIS Product Overview*, Santa Clara, California, 1991.

[12] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Henessey. The Stanford FLASH Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture,* Chicago, Illinois, April 1994.

[13] A. M. Mainwaring and D. E. Culler. Active Messages: Organization and Applications Programming Interface. *http://now.CS.Berkeley.EDU/Papers/am-spec.ps,* 1995.

[14] Meiko Scientific, ELAN Communications Processor Reference Manual, 1993.

[15] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing '95,* San Diego, California, 1995.

[16] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Typhoon and Tempest: User-Level Shared Memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture,* pages 325-336, Chicago, Illinois, April 1994.

[17] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 15th Annual Symposium on Operating System Principles,* pages 40 - 53, Copper Mountain Resort, Colorado, December 1995.

[18] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th Annual International Symposium on Computer Architecture,* pages 256-266, Gold Coast, Australia, May 1992.

[19] M. Welsh, A. Basu, and T. von Eicken. Low-Latency Communication over Fast Ethernet. In *Proceedings of EUROPAR '96,* Lyon, France, August 1996.

[20] M. Welsh, A. Basu, T. von Eicken. A Comparison of Fast Ethernet and ATM for Low-Latency Communication. In *Proceedings of the 3rd International Symposium on High-Performance Computer Architecture,* San Antonio, Texas, February 1997.