Improving the network interfaces for gigabit ethernet in clusters of PCs by protocol speculation

Author(s):
Kurmann, Christian; Müller, Michel; Rauch, Felix; Stricker, Thomas M.

Publication Date:
2000

Permanent Link:
https://doi.org/10.3929/ethz-a-006653917 →

Eidgenössische Technische Hochschule Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Departement of Computer Science *Laboratory for Computer Systems*

Improving the Network Interfaces for Gigabit Ethernet in Clusters of PCs by Protocol Speculation

Christian Kurmann, Michel Müller, Felix Rauch and Thomas M. Stricker

Laboratory for Computer Systems
Swiss Federal Institute of Technology (ETH)
CH-8092 Zürich, Switzerland
{kurmann,rauch,tomstr}@inf.ethz.ch
`http://www.inf.ethz.ch/`

CS Technical Report #339

April 27, 2000

Summary

Modern massively parallel computers are built from commodity processors and memories that are used in workstations and PCs. A similar trend towards commodity components is visible for the interconnects that connect multiple nodes in clusters of PCs. Only a few years ago the market was dominated by highly specialized supercomputer interconnects (e.g. in a Cray T3D). Todays networking solutions are still proprietary but they do connect to standard buses (e.g. as PCI card). In the future the networking solutions of the Internet (e.g. Gigabit Ethernet) will offer Gigabit speeds at lower costs. Commodity platforms offer good compute performance, but they can not yet fully utilize the potential of Gigabit/s communication technology, at least as long as commodity networks like Ethernet with standard protocols like TCP/IP are used. While the speed of Ethernet has grown from 10 to 1000 Mbit/s the functionality and the architectural support in the network interfaces has not kept up and the driver software becomes a limiting factor.

Network speeds are catching up rapidly to the streaming speed of main memory. Therefore a true "zero-copy" network interface architecture is required to sustain the raw network speed in applications. Many common Gigabit Ethernet network protocol stacks are called "zero-copy" at the OS level, but upon a closer look they are not really "zero-copy" down to the hardware level, since there remains a last copy in the driver for the fragmentation/defragmentation of the transfered network packets that are smaller than a page size.

Defragmenting all the packets of various communication protocols correctly in hardware remains an extremely complex task, resulting in a large amount of additional circuitry to be incorporated into to existing commodity hardware. Therefore we consider the different route of studying and implementing a speculative defragmentation technique, that can eliminate the last defragmenting copy operation from zero-copy TCP/IP stacks on existing hardware. The speculative technique shows even greater potential for improved efficiency once the present network interfaces are enhanced by a few protocol matching registers with a simple control path to the DMA engines.

The payload of fragmented packets is separated from the headers and stored into a memory page that can be mapped directly to its final destination in user memory. The checks for correctness and compliance with the protocol are deferred until, after several packets, an interrupt for protocol processing is taken. The success of a speculative approach suggests that a modest hardware addition to a current Gigabit Ethernet adapter design (e.g. the Hamachi chip) is sufficient to provide a high speed data path for zero-copy bulk transfers.

For an evaluation of our ideas we integrated a network interface driver with speculative defragmenting into existing zero-copy protocol stacks with page remapping, fbufs or user/kernel shared memory. Performance measurements indicate that we can improve performance over the standard Linux 2.2 TCP/IP by a factor of 1.5–2 for uninterrupted burst transfers. Based on those implementations we can present real measurement data on how a simple protocol matching hardware could improve the performance of bulk transfers with a commodity Gigabit Ethernet interface.

As with any hardware solution using speculative techniques, a fairly accurate prediction of the good case (i.e. that a sequence of incoming packets are consecutive) is required. We show success rates of uninterrupted bulk transfers for a database and a scientific computation code on a cluster of PCs. The hit rate can be greatly improved with a simple matching mechanism in the network interface that allows to separate packets suitable to zero-copy processing from other packets to be handled with a regular protocol stack.

1 Introduction

Data rates in the Gigabit/s range are one of the enabling technologies for collaborative work applications like multimedia collaboration, video-on-demand, digital image retrieval or scientific applications that need to access large data sets over high speed networks. Unlike conventional parallel programs, these applications are not coded for APIs of high speed message passing libraries, but expect the standard socket API of a TCP/IP protocol stack.

1.1 State of the Art in Gigabit Networking

Over the past five years several different Gigabit/s networking products for clusters of PCs were announced. Three prominent examples are Gigabit Ethernet (1000BaseSX) [22], Myrinet [2] and the Scalable Coherent Interconnect (SCI) [16] that can connect to any workstation or PC through the PCI bus. Although Gigabit networking hardware is readily available the discrepancy between hardware performance and overall system performance including driver software remains higher than with the other technologies. The installation of a Gigabit/s network hardware in clusters of PCs often results in disappointing communication performance, especially if standard protocols like TCP/IP are used. Figure 1 shows the data rates achieved for large transfers with several known TCP protocol stacks and, as a reference with the best possible speed, for the French BIP MPI message passing libraries [21]. Both tests execute over our two favorite Gigabit/s networks interconnecting the same type of PC hardware.

The MPI performance figures close to the hardware limit prove that data transfers at a Gigabit/s speed can indeed be done even with commodity platforms with a PCI bus based network interface card.

Our Gigabit Ethernet test bed comprises a SmartSwitch 8600 manufactured by Cabletron and 16 GNIC-II Gigabit Ethernet interface cards manufactured by Packet Engines plugged into high end Intel-based PCs. The state of the art TCP/IP stack implementation in Linux 2.2 is based on a *one copy* buffer management strategy. We measured a performance of 42 MByte/s for Linux and just 16 MByte/s for the same protocol in Windows NT.

1.2 Special Challenges of the Gigabit Ethernet Technology

Gigabit Ethernet, like all previous versions of Ethernet is designed for an unacknowledged, connectionless delivery service. A single frame can contain from 46 to 1500 Byte of data. The physical layer of

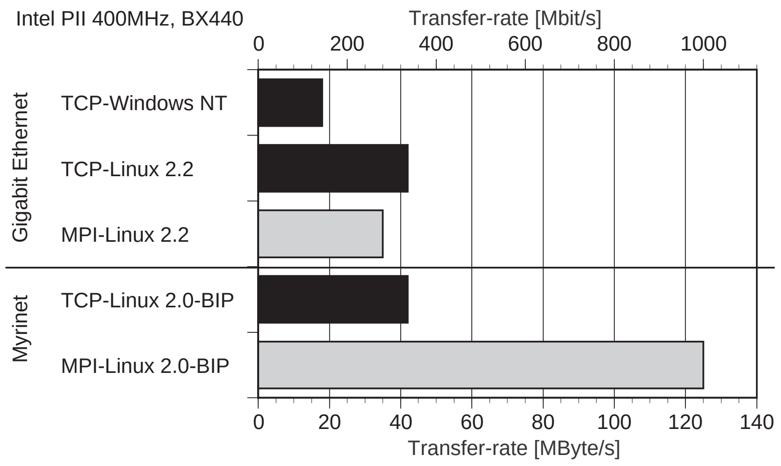

Figure 1: Throughput of large data transfers over a Gigabit/s point-to-point link with standard and special purpose networking technology and standard vs. specialized communication system software. All figures characterize the performance on 400 MHz Pentium II PCs with Intel 440 BX chipset.

switched, full-duplex Gigabit Ethernet, as considered in this paper, works with dedicated optical point-to-point connections supporting bi-directional communication with link-layer flow control as standardized in IEEE 802.3x. One of the big advantages of Gigabit Ethernet is the backward compatibility with its predecessor, the Fast Ethernet (100 Mbit/s). Despite recent improvements in speed, the maximal transfer unit (MTU) of Ethernet remains smaller than the page size of any operating system. Thus the current standards prevent page mapping from being used as copy technique at the driver level, unless larger IP packets can directly be fragmented in the driver. Therefore most Ethernet driver systems use one copy at the driver level.

Looking at the demands of error and congestion control mechanisms of Gigabit Ethernet, it might look completely hopeless to implement a fully compatible, fast zero-copy TCP/IP protocol stack with the existing simple PCI bus based network adapters. Full TCP/IP protocol support in hardware seems to be required to solve all potential problems with error and congestion control by means of retransmissions and exponential backoff. Looking more closely at the hardware solutions currently available things do look quite a bit better than expected. Based on link level flow control, a dedicated network control architecture for high speed communication can be devised and under certain assumptions, a rigorous zero-copy protocol is feasible and constitutes a big improvement for the communication patterns that make up most of the traffic in clusters of PCs.

In this paper we enlist techniques of speculation for efficient packet defragmentation at the driver level for TCP/IP over Ethernet. This technique makes it possible to use optimized zero-copy software architectures. Without changing its API, we furthermore optimized the implementation of the Linux TCP/IP stack and the socket interface which are used by many networking applications.

The idea of speculation immediately raises the issues of improved hardware support for better prediction or for better speculation about how to handle incoming packets most efficiently. We propose an approach where network adapters include simple protocol match CAM (content addressable memory) registers that classify incoming packets into at least two different categories: high speed zero-copy traffic and regular TCP/IP traffic.

The paper is structured as follows: After an overview of related work in zero-copy software and hardware architectures in Section 2 we describe a software only solution that provides the qualitative and the quantitative foundation for our improved Gigabit Ethernet network architecture targeted towards compute clusters in Section 3. Along the line of the software implementation we present the critical features of current network interface chipsets. The software only solution is based on speculation and tries to handle the common case fast at the cost of an expensive cleanup when things go wrong. In Section 4 we suggest ways to improve the likelihood of a fast transfer by adding multiple descriptor lists and a selecting match cam register to separate fast protocol traffic from regular traffic. In Section 5 we briefly discuss the role of a dedicated network control architecture for cluster computing that can take advantage of fast transfers. Finally in Section 6 we present some highly encouraging performance results to quantify the success rates, the best case as a benefit of successful speculation and the penalties of cleanup when speculation fails. In Section 7 we conclude.

2 Related Work in the Area of Zero-Copy Communication

Previous work resulting from the early projects in network computing [15] established the fact, that badly designed I/O buses and slow memory systems in PCs or workstations are the major limiting factor in achieving sustainable Gigabit/s speeds in communication services for application programs. Today's high speed networks, the I/O systems and the hierarchical memory systems all operate at a comparable bandwidth of about 100 MByte/s. Therefore the remaining and most important challenge for communication system software is to *prevent data copies*.

In the next paragraph we will give an overview of zero-copy software techniques and networking technology alternatives which provide optimized hardware support.

2.1 Zero-Copy Software Architectures

Recent efforts have been focused on designing optimized software architectures, called *zero-copy*, capable of moving data between application domains and network interfaces without CPU and memory bus intensive copy operations. A variety of approaches to host interface design and supporting software have been proposed. To give an overview of previous and related work we slightly extend a classification in [6].

1. **User Accessible Interface Memory**: A scenario with minimal data transfer overhead is one in which the network interface memory is accessible and pre-mapped into the user and kernel address space (variation described in [7]). This approach requires complicated hardware support and substantial software changes.

2. **User-Level Network Interface (U-Net) or Virtual Interface Architecture (VIA)**: Another alternative is to choose a low level hardware abstraction for the network interfaces and leave the implementation of a suitable communication system software to libraries in user space [23, 11, 10]. This direct access to the network interface leads to very low latencies and it allows a large flexibility in the choice of the protocol, but some considerable hardware and software support is required for network interface cards with multiple virtual interfaces.

3. **User/Kernel Shared Memory**: This scheme defines a new set of APIs with shared semantics between the user and kernel address space and uses DMA to move data between the shared memory and network interface. Proposals in this category are *fast buffers (fbufs)* [9] and *IO-Lite* [20]. It uses per-process buffer pools that are pre-mapped in both the user and kernel address spaces, thus eliminating the user-kernel data copy.

4. **User/Kernel Page Remapping with Copy on Write**: This approach re-maps memory pages between user and kernel space. At the sender the user data is mapped to the kernel space where the

corresponding protocol headers are generated. The scheme then uses DMA to transfer the frames between kernel buffers and the network interface card. At the receiver buffers are re-mapped from kernel space to user space by editing the MMU table. An implementation for TCP/IP over ATM on Solaris is described in [6].

Despite a strict zero-copy handling within the operating system, the implementation techniques mentioned above still fail to remove the last copy in the Gigabit Ethernet driver that is due to packet defragmentation [1]. On the other hand the idea of manipulating the behavior of the Ethernet driver to reduce in-memory copy operations is not new and it has been explored with conventional Ethernet at 10 MBit/s. The performance improvement was modest at the time since in memory copies were faster and less critical at 10 MBit/s speeds.

5. **Blast transfer facilities**: A special transfer protocol is used for large transfers and during such a transfer the driver's buffer chain is changed in a way that the headers and the data of the incoming packets are separated by the network interface card. In [4] the data parts are directly written to user space, while [19] re-maps the pages with the contiguous data parts from kernel to user space like it is done in our implementation. The headers go to kernel buffers in both cases. These schemes need to take some special action in case the blast transfer is interrupted by some other network packet: The first approach is to limit the blast transfer length and copy the data to the correct location after an interrupted blast.

Since the investigations on blast transfers [4, 19] the network and memory bandwidths have increased by two orders of a magnitude, but the architecture of popular network interfaces for Fast and Gigabit Ethernet has hardly changed. Blast transfers use dedicated protocols, are limited to existing hardware and did not show as much improvement at their time (10 years ago). Therefore none of the blast techniques made its way into current clusters of PCs. Our work aims at changing this by improving the integration of fast transfers into existing protocol stacks. We also add the viewpoint of protocol speculation, that will clearly lead to a better understanding of the hardware support and that will point us towards simple and effective enhancements of the current network adapters.

2.2 Gigabit Networking Alternatives and their Solution for Zero-Copy

For System Area Networks (SAN) needed in clusters of PCs, there are a few highly attractive SAN alternatives to Gigabit Ethernet, among them are Myrinet [2], SCI [8], Giganet [13] and ATM-OC12. There are a few highly relevant differences of the Gigabit Ethernet vs. these other technologies. In all the other technologies the transfers to and from the host and the application program can be done in blocks equal or larger than a page size. This is important for zero-copy strategies.

Myrinet interconnects do feature long packets (unlimited MTU), link level error and end-to-end flow control that is properly handled by deadlock free routing in the wormhole switches. Furthermore the Myrinet network interface card does provide significant processing power through a user programmable RISC core with up to 2 MByte staging memory. With Myrinet, there is no justification for a complete TCP/IP protocol stack nor for breaking down IP packets into smaller units.

Similarly the bulk data transfers of SCI interconnects rely on error and flow control in the network to avoid the problem of fragmenting packets. Giganet also incorporates a supercomputer-like reliable interconnect and pushes a hardware implementation of VIA to provide zero-copy communication between applications from user space to user space. In contrast to these supercomputing technologies the ATM-OC12 hardware operates with packet losses in the switches and highly fragmented packets (53 Byte) just

[1]There are many successful prototypes, but most of them include semantic restrictions for zero-copy buffer management. A good overview is given in [3]. A better operating system support for I/O streams is described in [18].

like Ethernet. However with links that fast and packets that small, there is little hope for defragmentation in software and ATM adapters must provide this functionality entirely in hardware (e.g. for Solaris and ATM [6]). The rich ATM functionality comes at a certain cost, that might be too high for most PC clusters.

A similarly expensive hardware solution is SiliconTCP^{TM} [14] of InterProphet Corporation. According to their product information the maximal bandwidth achievable is 9 MByte/s, although these speeds are sustainable at a spectacularly low main processor utilization of only 2%. Still for Gigabit Ethernet an improvement of a factor of 10 in throughput would be required to deliver true Gigabit/s speeds with a TCP stack in hardware.

To overcome the problem of packets that are smaller than a memory page, some Gigabit Ethernet vendors propose a change of the standard by introducing Jumbo Frames [1]. In this solution the maximal Ethernet packet size (MTU) is increased to 9000 Byte and a proprietary network infrastructure supporting them is required. Nevertheless, the concept of Jumbo Frames does not solve the problem of header/payload separation and adversely affects the idea of using mainstream networking technologies in our clusters of PCs. Furthermore, many high-end SAN interconnects used wormhole forwarding in the past, where most LAN Ethernet switches are no longer reluctant to use the simple store-and-forward packet handling. In store and forward networks, high fragmentation and small packets translates into low latency.

Instead of demanding larger packet sizes we suggest to incorporate a very modest additional hardware support, enlist techniques of speculation and to go the route of efficient fragmentation at the driver level as this is already properly defined by the IP over Ethernet standard.

3 Enabling Zero-Copy for Existing Simple Ethernet Hardware

To achieve networking performance between 75 and 100 MByte/s with a standard Gigabit/s network interface a zero-copy protocol architecture is absolutely necessary. With the common restriction of only 1500 Byte as MTU (less than a page size) and the usual simple hardware support of descriptor based Ethernet network interfaces, it seems to remain impossible to solve the problem of a true zero-copy transfer because simplistic hardware cannot even separate protocol header and payload data.

To overcome the restrictions given by standard network technologies and the simple hardware functionalities, we propose to use *speculation techniques* in the receiver driver to defragment IP-packets with optimal hardware support. In our implementation of IP over Ethernet the driver pretends to support an MTU of an entire memory page (4 KByte) and handles the fragmentation into conventional Ethernet packets at the lowest possible level. By speculation we just assume that during a high performance transfer burst all Ethernet packets will arrive free of errors and in the correct order so that the receiving driver can put the payload of all fragments directly into the final destination. Once the defragmentation problem is solved and the packet is properly reassembled in a memory page all known zero-copy techniques can be used to move the payload to its final destination in memory. The checking of the proper order and the integrity of the packets received is deferred until a few packets are transferred and an interrupt for protocol processing is taken by the main processor.

As with all speculative methods, the aim is to make the best case extremely fast at the price of a potentially more expensive cleanup operation if something went wrong. We will show that the best case is indeed the common case, or that it will even become the most common case after a small amount of hardware is added to the network interface design.

The speculative defragmenter runs in parallel to a regular TCP/IP stack. If the data is not fragmented or optimistic zero-copy defragmentation does not succeed the data is passed to the regular protocol stack to be handled in a conventional one- or two-copy manner. Fragmenting Ethernet packets at the driver level also needs support at the sender side, but such is simple to provide. A speculative defragmenter can run with simple descriptor based hardware, i.e. the addresses of the incoming packets are preprogrammed. Only one high performance stream can processed at one time with good performance. Although zero-copy transfers must be scheduled explicitly or implicitly between two nodes they can coexist with normal networking traffic. At this point we are able to show the performance of the speculative defragmenter based on a software implementation, but also suggest some simple packet filtering hardware to improve the odds of speculation significantly if multiple streams are to be processed.

3.1 Gigabit Ethernet and its NICs

In our experimental cluster of PCs we use off-the-shelf 400 MHz Pentium II PCs, running under a Linux 2.2 kernel, connected via Gigabit Ethernet by fibers. Gigabit Ethernet, as recorded in the 802.3z standard [22] is the latest speed extension of the well known Ethernet technology. It is successful for several reasons. The technology is simple, which translates to high reliability and low maintenance cost as well as a reasonable cost of entry (e.g. compared to ATM). Gigabit Ethernet is layered on top of the already developed and tested physical layer of enhanced ANSI standard Fiber-Channel optical components that are well proven for connecting a network host-adapter to a central high performance packet switching backplane (i.e. IEEE 802.3).

3.1.1 The Hamachi Ethernet Interface Chipset

The Hamachi chipset is a typical Gigabit Ethernet controller. Besides some buffering FIFOs towards the link side the controller chip hosts two independent descriptor-based DMA processors (TX and RX) for streaming data to and from host memory without host intervention, hereby reducing the CPU load necessary for network handling. Advanced interrupt coalescing techniques reduce the number of host interrupts to a minimum and multiple packets can be handled with a single interrupt. The controller chip also detects TCP/IP protocol frames and correctly calculates the necessary checksums while forwarding the packets to the host. Some large internal FIFOs and an extended buffering capability in external SRAM chips maximize the autonomy of operation and limit the chances of packet loss when the host processor is heavily loaded.

3.2 An Implementation of the Zero-Copy TCP/IP Stack

For a prototype implementation of a zero-copy TCP/IP stack with driver level fragmentation we use several known techniques as indicated in Section 1 — in particular “User/kernel page remapping with copy-on-write” as in [17] and alternatively fast buffer “fbufs” concept as proposed in [5]. The two different OS implementations demonstrate that the speculative defragmentation technique works with different OS embeddings.

3.2.1 Speculative Defragmentation with the Hamachi Chipset

Our fragmenting Ethernet driver manages to send an entire 4 KByte page across a network with an MTU of only 1500 Byte (payload). A 4 KByte zero-copy packet is decomposed into three fragments, each fragment using two DMA descriptor entries — one for the header data and one for the application data. Therefore six descriptors are used to transmit or receive one fragmented zero-copy packet. Without parameter tuning TCP allows to send a maximum of 64 KByte at a time in a window, which corresponds to 16 zero-copy packets, so at least 96 descriptors are required to handle a burst. This scheme of descriptor management permits to use the DMA-engine of the NIC in order to fragment and defragment packages directly from and into memory pages that are suitable for mapping between user and kernel space.

A descriptor entry comprises fields for status, a length of the transfer and a pointer to the data in the buffer. The Hamachi controller indicates with an *End_Of_Packet (EOP)* flag whether the bytes of an incoming frame extend across several buffers. Thanks to this indicator it becomes possible to automatically separate headers from payload. The problem is that descriptors must be statically pre-loaded in advance and that a proper separation of header and data is speculating on the length of the header. For the zero-copy implementation the IP- or TCP-options fields must be pre-negotiated[2]. Should an incoming frame not match the expected format of an IP fragment, or if it contains unexpected protocol options, the zero-copy handling is aborted and the packet would have to be passed to the regular protocol stack and copied. In the present implementation every unexpected non-burst packet causes zero-copy processing to be disrupted. Figure 2 shows a snapshot of a descriptor list after the buffers were written by the DMA.

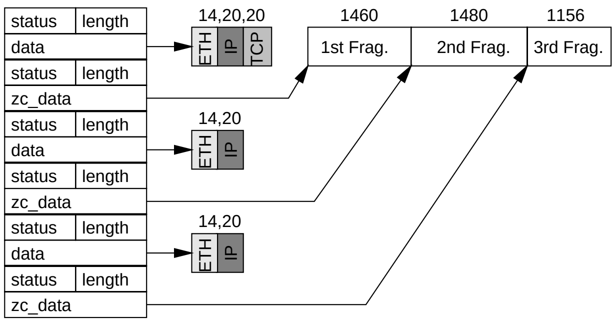

Figure 2: Descriptor list with 6 entries showing a defragmented 4 KByte packet. The header data consists of an Ethernet and an IP part, in the first packet additionally a TCP part.

3.2.2 Packet Transmission

In place of the standard IP stack our NIC driver will take on the responsibility of fragmenting packets larger than 1500 Byte (payload) into appropriate fragments. The driver simply emulates a virtual network interface that allows to send Ethernet frames of 4 KByte in size. The fragmentation is done in a standard way by setting the *more fragments* flag in the IP-header and by calculating the proper offsets within the original packet as outlined in Figure 3. The status of transfer (fast or regular) and packet length is known to the sender, so there is no speculation involved here.

3.2.3 Packet Reception

Upon packet arrival the Hamachi controller logic will transfer the header and the payload into the buffers in host memory designated by the pre-loaded receive descriptor list. After all the fragments are in or alternatively after a timeout occurs and an interrupt is triggered, the Ethernet driver will check whether the payloads of all received fragments have been correctly written to the corresponding buffer space. Those checks involve the protocol family, the IP-packet IDs and the fragment offsets. If everything went well, the IP-header is adapted and the 4 KByte packet passed to the IP-stack. If an interfering packet was received by the driver, the data is copied out of the wrongly assumed final location into a normal socket buffer and passed to the IP-stack for further processing.

[2]The Hamachi chip does protocol interpretation to determine the payload for checksumming, but we found no way to use this information to control the DMA behavior. We hope that future Gigabit interface designs have better support for this.

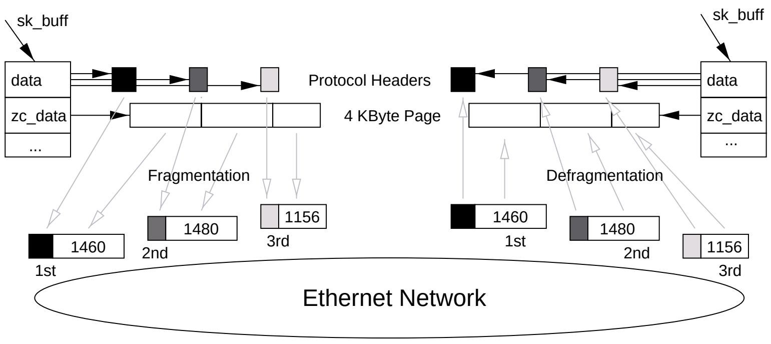

Figure 3: Fragmentation/Defragmentation of a 4 KByte memory page is done by DMA through the interface hardware.

3.2.4 Resynchronization

If the zero-copy data stream is disturbed by an unexpected packet, the receiver has to abort zero-copy processing and re-synchronize itself. As soon as the device driver detects this condition the receiving DMA is stopped and all the packets thereafter are copied into buffers and passed to the regular IP-stack. Hence back pressure is applied to the link and a packet loss can be prevented in most cases due to buffers in the NIC and the Gigabit Switch. After cleanup is complete the DMA is restarted by a simple write to a PCI mapped register on the Hamachi controller.

4 Enhanced Hardware Support for Speculative Zero-Copy

In Section 3.2 we showed that, based on speculation techniques, it is possible to implement a good zero-copy TCP/IP protocol stack with the simple hardware support available in today's Ethernet network interfaces. The speculative technique required extensive cleanup operations to guarantee correctness in all cases, when the defragmenter made wrong guesses about the packet order. Speculation misses automatically raise the question of better prediction hardware to improve the accuracy of the guesses. We therefore propose a simple extension to the NIC hardware that could greatly simplify the implementation of a zero-copy defragmenter, while preserving the compatibility with standard Ethernet IP protocols and the simplicity of off-the-shelf Ethernet hardware.

4.1 Multiple Descriptor Lists for Receive

Communication at Gigabit/s speeds requires that incoming data is placed automatically into the memory of the receiving processor. At Gigabit/s speeds there is no time to take an interrupt after every incoming packet or to do a complete protocol interpretation in hardware. For zero-copy the difficult part is to deposit incoming, possibly fragmented payload directly into its final destination in memory. The previous work with ATM or VIA [23, 11] suggests that this is done on a per virtual channel or per connection basis. All known solutions require lots of dedicated hardware, a co-processor or copies to solve the problems.

Based on our strategy of speculative support we propose to divide the available descriptors into a small number of separate lists to deposit different kinds of incoming data segments directly. We suggest one list for transmitting and at least two descriptor lists for receiving, one that is usable to handle fast data transfers in a speculative manner with zero copy and a second one that can handle the packets of all other traffic including all unexpected packets in a conventional manner. The small increase from currently two to three descriptor lists satisfies our requirements for a minimal change to the current adapter design.

4.2 Content Addressable Protocol Match Registers

The goals of using standard Ethernet switches in cluster computing is to work with mass market standard equipment for the interconnects and the switches and therefore the standard Ethernet and IP protocols must be followed as closely as possible. Messages that are handled in a zero-copy manner must be tagged appropriately. Zero-copy messages could be tagged at the Ethernet frame level, with a modified protocol ID, but this could cause problems with smart switches. We decided to use the *Type-of-Service* bits within the IP header.

Many networking adapters do already protocol detection to help with the checksumming of the payload but this detection information can not be used to control the transfer of the frames to the host. Also a built in detection is static and not programmable to a specific data stream. To keep our protocol detection hardware as flexible and as simple as possible, we suggest to include a user programmable Match CAM (Content Addressable Memory) register of about 256 bit length for every descriptor list. The first 256 bits of any incoming packet are matched against those registers and the match is evaluated. The CAM properties of the match register are tailored to interpretation of protocols modified in the future and of entire protocol families. The bits should be maskable with a “don’t care”-option and it would make sense to provide some modest associativity (e.g. 2way or 4way) for matches. An Example of a packet match is given in Figure 4.

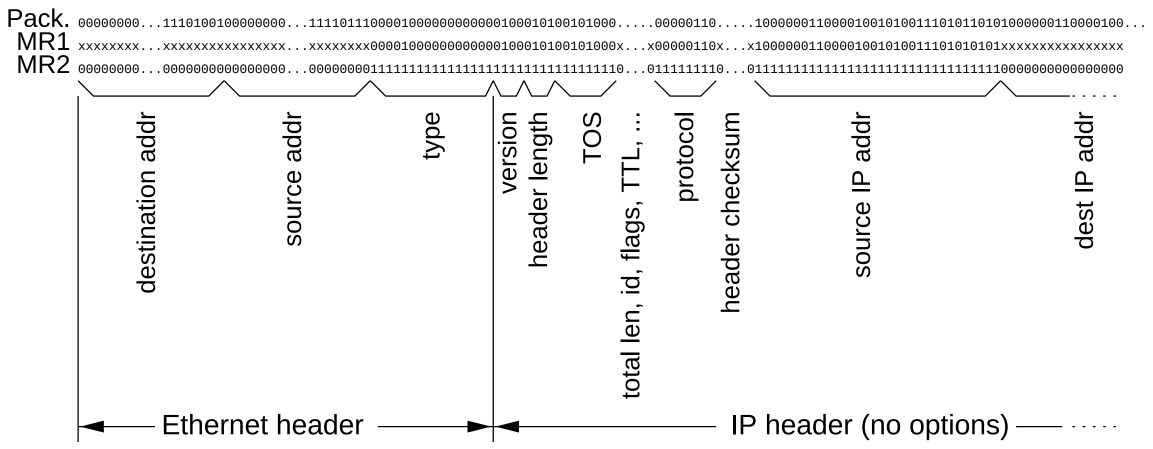
Pack. 00000000...1110100100000000...1111011100001000000000000010001010010100 0.....00000110.....10000001100001001010011101011010100000011000010 0...
MR1 xxxxxxxx...xxxxxxxxxxxxxxxx...xxxxxxxx00001000000000000010001010010100 0x...x00000110x...x100000011000010010100111010101 01xxxxxxxxxxxxxxxx
MR2 00000000...0000000000000000...000000001111111111111111111111111111111111 0...0111111110...011111111111111111111111111111111110000000000000000
destination addr
source addr
type
version
header length
TOS
total len, id, flags, TTL, ...
protocol
header checksum
source IP addr
dest IP addr
Ethernet header
IP header (no options)

Figure 4: The bit stream of every incoming packet is matched against a content addressable match register (match cam) to detect packets to be handled by the zero-copy defragmenter. Matching against an “x” (don’t care) in the match register is implemented with a second mask register. Multiple match register pairs provide associativity to match more than one protocol family. The match registers are mapped into the control space of the NIC and can easily be written by the drivers.

The best location of this protocol matching hardware strongly depends on the VLSI implementation of the Gigabit Ethernet interface, as shown in Figure 5. In the most simple FIFO based designs the match operation against the first 256 bits of an incoming packet can take place at the head of the FIFO (1) while the data and the match results are propagated to the end of the queue. In the more advanced design with mandatory buffering into a staging store (like e.g. in the Hamachi chipset) the protocol match could also be evaluated when the packet is transferred into the internal staging SRAM (2) and again the result of the match would be stored with the packet. As a third option the match could be performed just before the DMA scheduler selects a packet for the transfer over the PCI bus into the host (3). The third option is probably too late since the DMA schedule needs to decide based on the availability of descriptors to transfer or not to transfer a packet to host memory. The preference of VLSI designers for early evaluation eliminates many speed path limitations, but can potentially cause coherence problems when the match registers are changed on the fly during network operations.

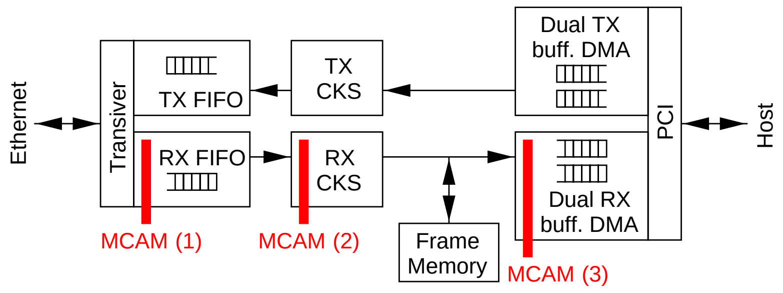

Figure 5: Schematic flow of the packets through a typical Gigabit Ethernet adapter. The gray bars indicate three possible locations (1), (2) or (3) for the proposed protocol match cam mechanism. We recommend the location (1) at the beginning of the RX FIFO since it provides the matching information as early as possible.

Match CAMs were extensivly used in the iWarp parallel computers to detect and separate messages of different protocol families in its high speed interconnect. The implementation in the iWarp VLSI component [12] consumed less than 2000 gates. Matching registers are already present in many NICs although they usually control checksumming logic and not the descriptor selection for DMA.

5 A Network Control Architecture

The problem with a speculative solution are multiple concurrent blast transfers to the same receiver, which could garble the zero-copy packets and reduce the performance because of frequent mis-speculation about the next packets. To prevent interfering packets, we implemented a transparent admission control architecture on the Ethernet driver level that promotes only one of the incoming transfer streams to fast mode. Unlike in blast facilities, this control architecture does not necessarily involve protocols different from regular IP or an explicit scheduling of such transfers.

While no precise knowledge of our dedicated network control architecture is required to understand the issues of hardware support for fast transfers, some knowledge about it might be beneficial for the interpretation of our performance results in the next Section.

5.1 Admission Control for Fast Transfers

One possibility to prevent interference with zero-copy fast transfers is to limit an end point to just allow one sender to blast a fast transfer. We have successfully implemented a distributed admission control mechanism on Ethernet driver level which achieves an exclusive allocation of a host-to-host channel. We therefore use especially tagged Ethernet packets which are handled directly by the NIC driver itself.

A sender sends a *Fast_Req* to the receiver which is answered by a *Fast_Ack* or a *Fast_NAck* packet. The round trip time of such a request is about 30 μs but does not delay the data transfer by the same amount as the data transmission can start immediately when the data is available to the driver. As soon as an acknowledgment arrives at the sender driver, all further packets are sent as fast packets (in our software implementation this simply means that the packets are sent as three especially fragmented IP packets of 4096 Bytes in total).

On arrival of a non-acknowledgment packet the data transfer is either stopped and delayed or just continued as a normal transfer with low bandwidth to reduce interference with the ongoing fast transfer of another sender. The non-acknowledge may contain the expected duration of the current fast transfer and thereby giving out a hint on when the channel may be deallocated by the fast transfer in progress. For low priority or regular transfers we use the standard flow control algorithm of TCP/IP to slow down the bandwidth by just dropping packets or delayed acknowledgments. We will show in the performance analysis that such low priority data-streams do not affect the bandwidth of a fast transfer in progress even in the software solution where a re-synchronization of the descriptors has to be done for each packet which interferes a fast transfer.

5.2 Implicit versus Explicit Allocation of Fast Transfers

With the above described scheme it is possible to hide the whole control architecture from the user program to optimize the end-user communication performance automatically. The operating system or the middleware knows the size of a transfer when the corresponding system call is called. If the transfer is potentially large enough for a fast transmission, the channel is then automatically requested, set up and receives the benefit of a highly increased bandwidth for data transfers.

As an alternative the selection of the transfer mechanisms can be put under application control. An application would only transfer the data if the fast channel were free and otherwise try to reschedule its transfers and try to use another fast channel that is available at the time. We therefore implemented an I/O-control (`ioctl`) call which allows to send requests and returns the answers of the receiver. As for many large parallel applications the communication pattern of large transfers is quite regular and scheduling is done by the programmer anyway, i.e. sending MPI messages, so this mode of resource allocation in the network interface may deliver the best results.

6 Performance Results

6.1 Performance Limitation in PCI based PCs

The two Gigabit/s networking technologies introduced earlier are able to provide at least a Gigabit/s transfer rate over the network wires. Therefore — in theory — a 1000BaseSX Ethernet allows transfer rates of about 125 MByte/s and a Myrinet of about 160 MByte/s.

The external PCI bus in current commodity PCs runs at 33 MHz with a 32 bit data path permitting data transfer rates of up to 132 MByte/s in theory. The maximal performance in practice is 126 MByte/s, as we measured with our PCI Myrinet interface cards for large burst transfers between the staging memory on the card and the main memory of the PC.

The performance for a memory to memory copy by the CPU is at 92 MByte/s for the Intel 440 BX chipset operating an SDRAM based memory system clocked with 100 MHz. Therefore a back-of-the-envelope calculation predicts a theoretical performance limit of 100 MByte/s for a zero-copy implementation, 50 MByte/s for a one-copy implementation and 33 MByte/s for a two-copy protocol stack.

6.2 Measured Best Case Performance with Zero-Copy

Distributed applications executing on top of the standard Linux 2.2 kernel achieve a transfer rate of about 42 MByte/s for large transfers across a Gigabit Ethernet (see Figure 6). With the speculative defragmentation alone (i.e. without a zero-copy embedding into an OS) the bandwidth increases to 45 MByte/s. The current implementation of the speculative defragmenter attempts to receive three packets in a row and therefore reduces the interrupt load on the receiver by delaying the generation of an interrupt until at least three packets have been received or a timeout is reached. The limit of three packets is considerable lower than the very large transfers considered in previous work about blast protocols. This improves the minimal length for half of peak speed (half-length) considerably.

After an integration of our defragmenting driver into the zero-copy OS environment the performance of the TCP/IP stack is increased from 42 MByte/s for standard TCP/IP under Linux 2.2 to 65 MByte/s for transfers in a "page remapping" environment and 75 MByte/s in a "fast buffers" environment. The "page remapping" approach is slightly slower than the "fast buffers" approach since with "fast buffers" the expensive memory mapping operations are performed in advance.

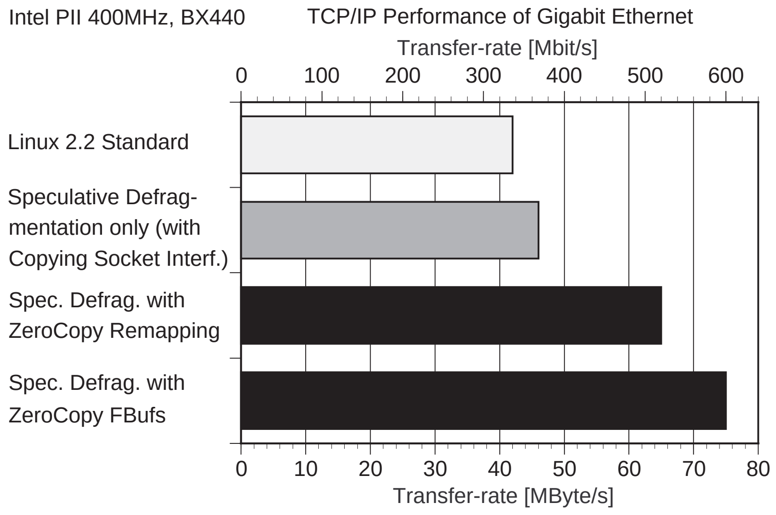

Figure 6: Throughput of large data transfers over Gigabit Ethernet (light grey bar). In combination with a zero-copy interface (fbufs) the throughput is increased from 42 MByte/s to 75 MByte/s (black bar). The speculative defragmentation alone does not increase the performance much (dark grey bar), since in the upper layers of the system, data is still copied between user and kernel space. The zero-copy remapping or fbufs without our optimized driver cannot achieve a satisfying performance, since data is still copied in the driver during defragmentation (not measured). Only the combination of the two techniques enables the higher speeds of true zero-copy communication (black bars). The figures characterize the performance on two 400 MHz Pentium II PCs with an Intel 440 BX chipset.

6.3 Performance of Fallback (Penalties when Speculation Fails)

A first fallback scenario investigates the performance of a sender that dumps fragmented 4 KByte TCP packets to an unprepared standard Linux receiver. As mentioned we use standardized IP-fragmentation and so every receiving protocol stack is able to handle such a stream without any problems at normal speed (see Figure 7). Actually the performance of this fallback scenario is higher than the standard protocol stack; the negligible improvement is probably due to the more efficient sender fragmenting in the driver rather than in the TCP/IP protocol stack.

The more interesting fallback case is when speculation in the driver fails entirely and a fallback with cleanup is required: If the device driver detects that a packet not belonging to the current stream was received as zero-copy packet, two actions need to be taken: (1) The reception of more packets must be stopped, the descriptor lists reinitialized and the reception of packets restarted, (2) the scattered fragments must be copied from the zero-copy buffers and passed on to the regular IP-stack. Both actions are fairly infrequent in a blast transfer even if the zero-copy transfer is interrupted with other transfers as seen in

Packets interfered	Failed ZC-Packets	Bandwidth [MB/s]
0	2	75
100	15	73
10000	28	63

Table 1: Effect of interferences on a transfer with the `ttcp` program and 100'000 zero-copy packets containing 4096 Byte.

Table 1. The maximum cost of this worst case is depicted with the case that a standard sender transmits to a receiver in zero-copy mode, as seen in Figure 7.

The overheads of a cleanup mentioned above reduce the performance from 42 MByte/s to about 35 MByte/s.

For pre-scheduled communication, the cleanup scenario is already highly unlikely and the probability of the occurrence can be substantially reduced by an appropriate network control architecture that coordinates the senders and receivers in a cluster (see Section 5). With the proposed simple packet filter through a match CAM register in the NICs (as described in Section 4) which would separate any burst streams from all the other traffic the miss rate could be further reduced.

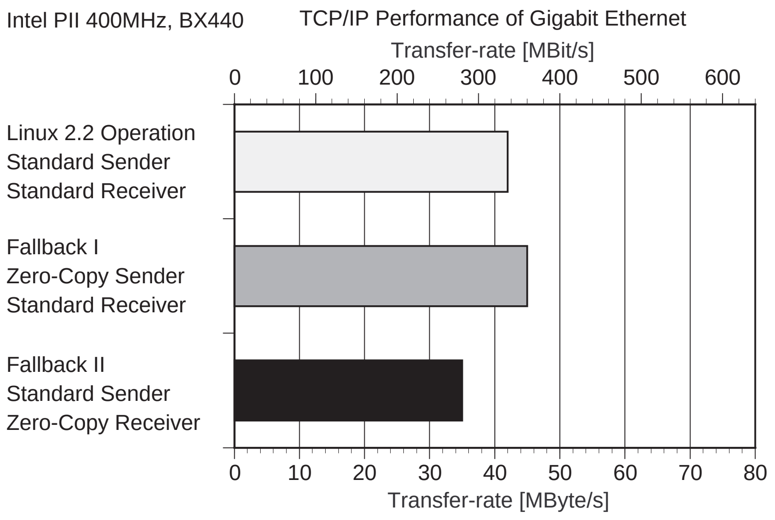

Figure 7: TCP throughput across a Gigabit Ethernet for two fallback scenarios. There is no penalty for handling sender fragmented packets at an unprepared receiver in normal receive mode. Only if a standard sender is interrupting a burst into into a zero-copy receiver, cleanup and resynchronization after each packet is needed. This case should be infrequent and remains unoptimized, it still performs at 35 MByte/s.

6.4 Rates of Success in Applications

6.4.1 Rates of Success for the Software-only Solution

Table 2 shows traces of two applications running on our cluster of commodity PCs. The first is a cluster of oracle databases executing the query 7 of a TPC-D workload (distributed across multiple nodes of the cluster by an SQL parallelizer) and the second is the execution of an OpenMP SOR code on a cluster

Application trace		Oracle running TPC-D			TreadMarks running SOR		
		Master	Host 1	Host 2	Master	Host 1	Host 2
Ethernet frames	total	129835	67524	62311	68182	51095	50731
	large (data)	90725	45877	44848	44004	30707	30419
	small (control)	39110	21647	17463	24178	20388	20312
ZC packets	potential	26505	12611	13894	14670	10231	10135
	successful	12745	12611	13894	14458	10225	10133
	success rate	48%	100%	100%	99%	>99%	>99%

Table 2: Study about the rate of success for speculative transfers based on application traces (numbers signify received frames/packets). The TreadMarks application prevents interferences of fast transfers whereas the TPC-D benchmark needs the control architecture to guarantee a predication rate that makes speculation worthwhile.

of PCs using the TreadMarks DSM system (distributed shared memory). Both applications have been traced for their communication patterns that show different results. In the Oracle case with the TPC-D workload, two bursts containing results from queries are simultaneously communicated back to the master at the end of the distributed queries. This leads to many interferences, which would be separated with additional hardware support as described in the previous Chapter.

In the TreadMarks example, pages are distributed over the network at the beginning of the parallel section and sent back to the master at the end of the parallel section. The structure of the calculations or the middleware properly serializes and schedules the transfers so that the speculation does work perfectly. There was no need for additional hardware support in this benchmark.

6.4.2 Rates of Success for the Hardware Enhanced Solution

A proper evaluation of our proposed hardware improvements remains extremely difficult for a university research group with a focus on software and entire systems. Unfortunately we can not engage into an ASIC or custom VLSI design to implement the proposed changes nor do we have access to the VHDL hardware description of the Hamachi chip set for simulation. Therefore a coarse grain behavioral model must suffice to evaluate the benefit of the design proposal.

With the multiple descriptor lists and the flexible matching mechanism the optimized zero-copy and standard streams will become cleanly separated and the fast stream can remain free from disturbances by other protocols. Still due to the properties of Ethernet the solution remains speculative, since there are no guarantees that the incoming stream is free of errors and free of packet losses. To keep probabilities of loss and cleanup down, all link and switch fabric flow control mechanisms of modern Gigabit Ethernet switches must be utilized.

This suggested simple hardware enhancement could greatly improve the chances for speculative defragmentation to succeed. For simple communication patterns (like point-to-point, one-to-many) the miss rates were found to be negligible. For complex and congested communication patterns the miss-rates have not been determined yet. However we have instrumented our software solution to collect statistics about packets that were wrongly speculated about. As we move to use our 16 node Gigabit Ethernet cluster of PCs in more realistic applications we might soon have a definitive, quantitative idea about hit and miss rates and the potential gain of the suggested hardware improvements.

7 Conclusions

The small packet size of standard Gigabit Ethernet prevents common zero-copy protocol optimizations unless IP packets or MPI messages can be fragmented most efficiently at the driver level without involving any additional data copies. Fragmenting and defragmenting packets correctly and accurately in hardware remains impossible with the existing Gigabit Ethernet interface chips unless a *speculative approach* is taken.

Our speculative packet defragmenter for Gigabit Ethernet successfully relies on an optimistic assumption about the format, the integrity and the correct order of incoming packets using a conventional IP stack as fallback. The driver works with the DMAs of the network interface card to separate headers from data and to store the payload directly into a memory page that can be remapped to the address space of the communicating application program. All checks whether the incoming packets are handled correctly according to the protocol are deferred until a burst of several packets arrived and if the speculation misses, some cleanup code passes the received frames to a conventional protocol stack for regular processing.

The idea of speculation immediately raises the issues of improved hardware support for better prediction or for better speculation about how to handle incoming packets most efficiently. A promising approach seems to include a few simple protocol match CAM (content addressable memory) registers that classify incoming packets into at least two different categories: high speed zero-copy traffic and regular TCP/IP traffic. The match cam registers control the selection of DMA descriptors used to store the packets in memory from different descriptor lists.

Our implementation of a network interface with a speculative defragmenter is embedded in an OS setting with common zero-copy techniques like “fast buffers” or “page remapping”. Together with those mechanisms a true zero-copy implementation of TCP/IP for Gigabit Ethernet has been implemented and measured. The implementation delivers 75 MByte/s transfers together with “fast buffers” support — a substantial improvement over the 42 MByte/s seen in the current standard TCP/IP stack in Linux 2.2 and 16 MByte/s measured with the corresponding stack for Windows NT 4.0 on top of a conventional NDIS driver.

We conclude that with speculation techniques it is indeed possible to implement true end-to-end zero-copy TCP/IP even with simple existing network adapters. A substantial fraction (60%) of the peak bandwidth of a Gigabit Ethernet can be achieved for large transfers while preserving standardized socket API and the TCP/IP functionality. The study of the speculation hit rates in two applications (SPLASH like FFT and a Oracle TPC-D benchmark) shows that misses are quite rare and that they can be further reduced by clever network control architectures.

We think that an enhanced implementation with better hardware support for speculation and fallback handler in software points towards a new direction for simple and efficient network interfaces in commodity systems. The promising performance results of our studies and implementations indicate that Gigabit Ethernet could soon become a viable interconnect for clusters of PCs.

References

[1] Inc. Alteon WebSystems. Jumbo frames. http://www.alteon.com/products/jumbo_frames.html.

[2] Nanette J. Boden, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet - A Gigabit per Second Local Area Network. In *IEEE Micro*, volume 15(1), pages 29–36, February 1995.

[3] J. C. Brustoloni and P. Steenkiste. Effects of buffering semantics on I/O performance. In *Proc. 2nd Symp. on Operating Systems Design and Implementation (OSDI)*, pages 277–291, Seattle, WA, Oct 1996. USENIX.

[4] John B. Carter and Willy Zwaenepoel. Optimistic Implementation of Bulk Data Transfer Protocols. In *Proceedings of the 1989 Sigmetrics Conference*, pages 61–69, May 1989.

[5] Irina Chihaia. Message Passing for Gigabit/s Networks with Zero-Copy under Linux. Master's thesis, Laboratory for Computer Systems, Swiss Federal Institute of Technology ETH, 8092 Zurich, Switzerland and Technical University of Cluj-Napoca, Romania, 1999. http://www.cs.inf.ethz.ch/.

[6] H. K. Jerry Chu. Zero-Copy TCP in Solaris. In *Proceedings of the USENIX 1996 Annual Technical Conference*, pages 253–264, San Diego, CA, USA, Jan 1996. The USENIX Association.

[7] Eric Cooper, Peter Steenkiste, Robert Sansom, and Brian Zill. Protocol Implementation on the Nectar Communication Processor. In *SIGCOMM90*, pages 135–143, Philadelphia, September 1990. ACM.

[8] Dolphin Interconnect Solutions. *PCI SCI Cluster Adapter Specification*, 1996.

[9] P. Druschel and L. L. Peterson. FBufs: A High-Bandwidth Cross-Domain Transfer Facility. In *Proc. Fourteenth ACM Symp. on Operating System Principles*, pages 189–202, Asheville, NC, December 1993.

[10] C. Dubnicki, E.W. Felten, L. Iftode, and Kai Li. Software Support for Virtual Memory-Mapped Communication. In *Proc. 10th Intl. Parallel Prof. Symp.*, pages 372–381, Honolulu, HI, April 1996. IEEE.

[11] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, 18(2):66–76, March-April 1998.

[12] T. Gross and D. O'Hallaron. *iWarp: Anatomy of a Parallel Computing System*. MIT Press, 1998.

[13] GigaNet Inc. http://www.giganet.com/.

[14] InterProphet. Silicontcp^{TM}. http://www.interprophet.com/.

[15] H.T. Kung, R. Sansom, S. Schlick, P. Steenkiste, M. Arnould, F. Bitz, F. Christianson, E. Cooper, O. Menzilcioglu, D. Ombres, and B. Zill. Network-Based Multicomputers: An Emerging Parallel Architecture. In *Proc. Supercomputing '91*, pages 664–673, Albuquerque, NM, Nov 1991. IEEE.

[16] Ch. Kurmann and T. Stricker. A Comparison of Three Gigabit Technologies: SCI, Myrinet and SGI/Cray T3D. In *SCI Based Cluster Computing, H. Hellwagner and A. Reinefeld, eds.* Springer, Berlin, Spring 1999. An earlier version appeared in Proc. of the SCI Europe'98 Conference, EMMSEC'98, 28-30 Sept 1998, Bordeaux, France.

[17] Michel Müller. Zero Copy TCP/IP for Linux with Gigabit Ethernet. Master's thesis, Laboratory for Computer Systems, Swiss Federal Institute of Technology ETH, 8092 Zurich, Switzerland, 1999. http://www.cs.inf.ethz.ch/.

[18] FW. Miller, P. Keleher, and SK. Tripathi. General Data Streaming. In *Proc. 19th IEEE Real-Time Systems Symposium*, pages 232–41, Madrid, Dec 1998. IEEE.

[19] Sean W. O'Malley, Mark B. Abbot, Norman C. Hutchinson, and Larry L. Peterson. A Transparent Blast Facility. *Internetworking: Research and Experience*, 1(2), December 1990.

[20] Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel. I/O-Lite: A Unified I/O Buffering and Caching System. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, pages 15–28, 1999.

[21] L. Prylli and B. Tourancheau. BIP: A New Protocol Designed for High Performance Networking on Myrinet. Technical report, LHPC and INRIA ReMaP, ENS-Lyon, 1997. http://lhpca.univ-lyon1.fr/.

[22] Rich Seifert. *Gigabit Ethernet: Technology and Applications for High-Speed LANs*. Addison-Wesley, May 1998. ISBN: 0201185539.

[23] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proc. of 15th Symposium on Operating Systems Principles (SOSP-15)*, Cooper Mountain, CO, USA, Dec 1995. ACM.