Cluster Computing 4, 7–18, 2001

Speculative Defragmentation – Leading Gigabit Ethernet to True Zero-Copy Communication

CHRISTIAN KURMANN, FELIX RAUCH and THOMAS M. STRICKER
Laboratory for Computer Systems, Swiss Federal Institute of Technology (ETH), CH-8092 Zürich, Switzerland

Abstract. Clusters of Personal Computers (CoPs) offer excellent compute performance at a low price. Workstations with "Gigabit to the Desktop" can give workers access to a new game of multimedia applications. Networking PCs with their modest memory subsystem performance requires either extensive hardware acceleration for protocol processing or alternatively, a highly optimized software system to reach the full Gigabit/sec speeds in applications. So far this could not be achieved, since correctly defragmenting packets of the various communication protocols in hardware remains an extremely complex task and prevented a clean "zero-copy" solution in software. We propose and implement a defragmenting driver based on the same speculation techniques that are common to improve processor performance with instruction level parallelism. With a speculative implementation we are able to eliminate the last copy of a TCP/IP stack even on simple, existing Ethernet NIC hardware. We integrated our network interface driver into the Linux TCP/IP protocol stack and added the well known page remapping and fast buffer strategies to reach an overall zero-copy solution. An evaluation with measurement data indicates three trends: (1) for Gigabit Ethernet the CPU load of communication can be reduced processing significantly, (2) speculation will succeed in most cases, and (3) the performance for burst transfers can be improved by a factor of 1.5–2 over the standard communication software in Linux 2.2. Finally we can suggest simple hardware improvements to increase the speculation success rates based on our implementation.

Keywords: zero-copy communication, networking, speculation, gigabit Ethernet, packet defragmentation

1. Introduction

High data rates in the Gigabit per second 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 for the socket API of a TCP/IP protocol stack.

The number of Fast Ethernet (100 MBit/s) and Gigabit Ethernet (1000 MBit/s) installations is rapidly growing as they become the most common means to connect workstations and information appliances to the Internet. High volumes translate into low unit costs and therefore Gigabit Ethernet technology could become highly interesting for cluster computing, although the technology itself was designed for a traditional networking world of globally interconnected networks (i.e., the Internet).

Over the past five years several different Gigabit/s networking products for clusters of PCs were announced. Three prominent examples of current Gigabit/s networking products are Gigabit Ethernet [23], Myrinet [3], and Scalable Coherent Interconnect (SCI) [17]. All three interconnects can connect to any workstation or PC through the PCI bus. Gigabit Ethernet networking hardware is readily available, but the discrepancy between hardware performance and overall system performance remains the highest for Gigabit Ethernet. For standard interfaces and standard protocols such as the socket API and TCP/IP the resulting communication performance is quite disappointing. Figure 1 shows the data rates achieved for large transfers with the message passing library BIP-MPI [12] and with standard networking protocol stacks TCP/IP as of fall 1999. The tests execute over Gigabit Ethernet and Myrinet interconnecting the same PC hardware. The Myrinet-MPI performance (lower gray bar) is close to the PCI bus limit and proves that data transfers at a Gigabit/s speed can indeed be achieved. The TCP performance (black bars) is about one third of the maximal achievable rate and illustrates the problem we are focusing on, the poor performance with a TCP standard interface over Gigabit Ethernet. One reason for the better performance of message passing over Myrinet the technologies large packet sizes and that there is no need for packet fragmentation.

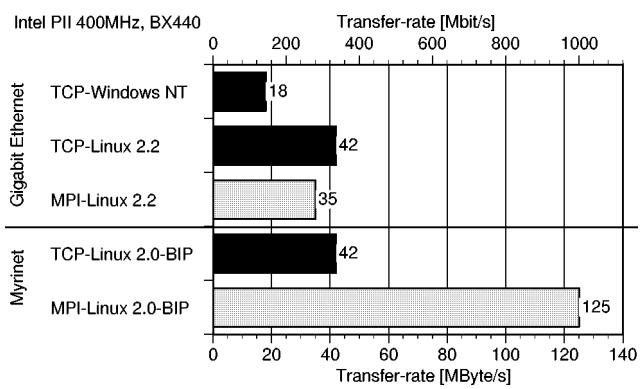

Figure 1. Throughput of large data transfers over Gigabit per second point-to-point links with standard and special purpose networking technology and communication system software.

Gigabit Ethernet, like all previous versions of Ethernet, has been designed for an unacknowledged, connection-less delivery service. The basic Gigabit Ethernet standard provides point-to-point connections supporting bi-directional communication including link-level (but no end-to-end) flow control. One of its big advantages is the backward compatibility with its predecessors, Fast Ethernet (100 MBit/s) and Classic Ethernet (10 MBit/s). The downside is that the maximum transmission unit (MTU) of Ethernet (1500 Byte) remains smaller than the page size of any processor architecture. So most Ethernet driver systems use at least one data copy to separate headers from data, rendering higher level zero-copy techniques useless, unless defragmentation can be done in hardware. The current Ethernet standard therefore prevent efficient zero-copy implementation.

Looking at the error recovery and the 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 network interface cards (NIC) for the PCI bus. The link-level flow control of Gigabit Ethernet can be extended with a dedicated network control architecture to enable high speed communication with a rigorous true zero-copy protocol under certain assumptions.

In the implementation described 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 highly optimized zero-copy software architectures. Without changing any API, we improved the Linux TCP/IP stack and the socket interface. The rest of the paper is structured as follows: section 2 gives an overview of related work in zero-copy software and hardware architectures, section 3 describes the speculation technique, that provides the qualitative and the quantitative foundation for our improved Gigabit Ethernet network architecture. Section 4 suggests ways to improve the likelihood for a successful speculative transfer with a dedicated network control architecture for cluster computing. Section 5 presents some highly encouraging performance results for synthetic benchmarks as well as execution times for applications. Finally, in section 6 we propose three small hardware changes that could greatly improve the success rate of speculation and simplify future driver implementations for zero-copy support.

2. Related work

Previous work resulting from early projects in network computing widely acknowledged that badly designed I/O buses and slow memory systems in PCs or workstations are the major limiting factor in achieving sustainable inter-node communication at Gigabit per second speeds [16].

Since today's high speed networks, I/O systems and hierarchical memory systems operate at a comparable bandwidth of roughly 100 MByte/s, one of the most important challenges for better communication system software is to *prevent data copies*.

In the next few paragraphs we give an introduction into previous work on zero-copy software techniques and into networking alternatives that are not subject to the problem of packet fragmentation due to their better hardware support.

2.1. Zero-copy software architectures

Many past efforts for better communication system software focused on designing so called *zero-copy* software architectures, that are capable of moving data between application domains and network interfaces without any CPU and memory bus intensive copy operations. A variety of host interface designs and their supporting software have been proposed. We adopt and extend a classification found in [7]:

1. *User-Level Network Interface (U-Net) or Virtual Interface Architecture (VIA)*: Those projects focus on low level hardware abstractions for the network interfaces and suggest to leave the implementation of the communication system software to libraries in user space [10,11,25].
2. *User/Kernel Shared Memory (FBufs, IO-Lite)*: Those proposed techniques rely on shared memory semantics between the user and kernel address space and permit to use DMA for moving data between the shared memory and network interface. The network device drivers can also be built with per-process buffer pools that are pre-mapped in both, the user and kernel address spaces [9,21].
3. *User/Kernel Page Remapping with Copy on Write and Copy Emulation*: These implementations re-map memory pages between user and kernel space by editing the MMU table and perform copies only when needed. They can also benefit from DMA to transfer frames between kernel buffers and the network interface [7].

There are many prototype implementations of the strategies mentioned above, but most of them include semantic restrictions of the API due to zero-copy buffer management or work only with network technologies supporting large frames. The semantic restrictions for zero-copy buffer management and zero-copy implementation basics was investigated in [4,5]. The authors of [19] describe a successful implementation of a standard communication library for cluster computing, strictly following the zero-copy paradigm and the authors of [18] propose better operating system support for I/O streams using similar principles.

Despite a strict zero-copy regime in the operating system, Gigabit Ethernet implementations cannot take full advantage of the techniques mentioned above, because they fail to remove the last copy due to packet defragmentation. The basic idea of modifying the Ethernet driver to reduce in-memory copy operations has been explored with conventional Ethernet at 10 MBit/s more than a decade ago in the simpler context of specialized blast transfer protocols.

4. *Blast transfer facilities*: Blast protocols improve the throughput for large transfers by delaying and coalescing acknowledge messages. The driver's buffer chain

is changed to permit the separation of headers and data of the incoming packets by the network interface card. In [6], the data parts are directly written to the user space, while in [20] the pages with the contiguous data are re-mapped from kernel to user space. The headers, on the other hand, go to kernel buffers in both cases. Both these schemes need to take some special actions in case the blast transfer is interrupted by some other network packet. An alternative approach limits the blast transfer length and copies the data to the correct location after an interrupted blast.

During the ten years since the investigations on blast transfers, the network and memory bandwidths have increased by two orders of a magnitude and different architectural limitations apply. However, the design of popular network interface cards for Fast and Gigabit Ethernet has hardly changed. Blast transfers used proprietary protocols and therefore none of the blast techniques made their way into cluster computing. So our work is quite different, since it tries to improve performance by an integration of fast transfers with existing hardware and protocol stacks and, as a novelty, we explore speculative protocol processing with cleanup code upon failure, while blasts work mostly with assertions in a deterministic way.

2.2. Gigabit networking alternatives and their solution for zero-copy

For System Area Networks (SAN) used in clusters of PCs several highly attractive SAN alternatives to Gigabit Ethernet are available, e.g., Myrinet [3], SCI [8], Giganet [13] and ATM-OC12. There are some relevant architectural differences between these interconnect technologies and Gigabit Ethernet. In some of those SAN technologies the transfers between the host and the application program can be done in blocks whose size is equal or larger than a memory page size; this is essential for zero-copy strategies and efficient driver software.

Myrinet interconnects support long packets (unlimited MTU), link level error detection and end-to-end flow control that is combined with a priori deadlock free, wormhole routing in the switches. Furthermore, the Myrinet network interface card provides significant additional processing power through a user programmable RISC core with large staging memory. With this hardware there is much less justification for a TCP/IP protocol stack. Similarly the bulk data transfer capability of SCI interconnects relies on its hardware for error correction and flow control 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 the supercomputing technologies, the ATM-OC12 hardware deals with packet losses in the switches and highly fragmented packets (MTU of 53 Byte). With fast ATM links and packets that small, there is no hope for defragmentation in software and the adapters must provide all functionality entirely in hardware. The rich functionality comes at a hardware cost, that proved to be too high for cluster computing. A similarly expensive hardware solution is SiliconTCP^{TM} [15]. The product implements a TCP/IP stack fully in hardware, which leads to the benefit of very low main processor utilization, but in its current implementation, fails to keep up with a Gigabit/s rate of fast interconnects.

2.3. Modifying the standard for Gigabit Ethernet

To overcome the problem of packets smaller than a memory page, some vendors changed the Gigabit Ethernet standard by introducing Jumbo Frames [1]. In this solution the maximal Ethernet packet size (MTU) is increased to 9 KByte and a proprietary network infrastructure supporting them is required. The deviation from the old Ethernet standard solve the problem of software defragmentation, but the concept of Jumbo Frames does not solve the problem of header/payload separation. Since most Ethernet switches use simple store-and-forward packet handling, storing packets in queues before being processing them and passing them on. Therefore in packet switched networks, the presence of Jumbo Frames will result in increased latencies for all packets traveling in the same network.

We go a different route – instead of demanding ever increasing packet sizes, we suggest to incorporate a modest additional hardware support for more fragmentation into smaller Ethernet packets (similar to ATM cells) which will result in lower overall latencies and still permit high bandwidth transfers. Since such interfaces are still not available today, we enlist techniques of speculation for efficient defragmentation at the driver level in software. This is foreseen and already properly specified by the classic Ethernet standard for IP traffic.

3. Enabling zero-copy for existing simple Ethernet hardware

In order to achieve networking performance between 75 and 100 MByte/s with a Gigabit per second network interface, a zero-copy protocol architecture is absolutely necessary. With the common restriction of the MTU to 1500 Byte (which is less than a memory page size) and the usual hardware support of a descriptor based Ethernet network interface, it seems impossible to solve the problem of a true zero-copy transfer, because the common simplistic DMA hardware cannot even reliably separate protocol header and payload data. To overcome the restrictions given by these standard network technologies and the simple hardware functionalities, we propose to use *speculative processing* in the receiving network driver to defragment IP-fragments with the current 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 assume that during a high performance transfer 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 well known zero-copy techniques can be used to pass the payload to the application.

As with all speculative methods, the aim of speculative defragmentation is to make a best case scenario extremely fast at the price of a potentially more expensive worst case scenario, with a cleanup operation if something went wrong. We will show by statistic arguments, that the best case is indeed the *most common* case. If either the data is smaller than a page or a speculative zero-copy defragmentation does not succeed because of interfering packets, the incoming data is passed to the regular protocol stack to be handled in a conventional one-copy manner. With the current hardware only one high performance stream can be processed at a time with optimal performance. On other network technologies, like, e.g., Myrinet, this restriction also holds since large transfers are non interruptible. Zero-copy transfers can be scheduled explicitly or automatically. A network control architecture with automatic scheduling is described in section 4. Both can coexist with normal networking traffic that is running without a performance penalty.

3.1. Gigabit Ethernet and its NICs

In our experimental cluster of PCs we use off-the-shelf 400 MHz Pentium II PCs, running Linux 2.2, connected via Gigabit Ethernet. Our Gigabit Ethernet test bed comprises a SmartSwitch 8600 manufactured by Cabletron and GNIC-II Gigabit Ethernet interface cards manufactured by PacketEngines.

Ethernet is highly successful for several reasons. The technology is simple, translating to high reliability and low maintenance cost as well as reasonable cost of entry (e.g., compared to ATM). Gigabit Ethernet was first layered on top of the already developed and tested physical layer of enhanced ANSI standard Fiber-Channel optical components and is now available over the Category 5 Copper cabling systems so that even the physical infrastructure of fast Ethernet can be reused.

Our NICs use the Hamachi Ethernet interface chipset that 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.

3.2. An implementation of a zero-copy TCP/IP stack with speculative defragmentation

For a prototype implementation of a zero-copy TCP/IP stack with driver level fragmentation we use several well-known techniques as indicated in section 2 – in particular, “page remapping” [7] and for a second alternative implementation the “fast buffer” concept proposed by [9]. The two different zero-copy techniques also demonstrate that the speculative defragmentation technique is an independent achievement and that it works with different OS embeddings.

3.2.1. Changes to the Linux TCP/IP stack for zero-copy

The Linux 2.2 TCP/IP protocol stack is similar to the BSD implementation and works with an optimized single-copy buffering strategy. Only a few changes were required to make it suitable for zero-copy. The only extension we added to the original socket interface is a new socket option allowing applications to choose between the new zero-copy socket interface and the traditional one.

In the original interface, the data to be sent is copied and checksummed out of the application buffer into a so called *socket buffer* in kernel space. Instead of copying the data, we just remap the whole page from user to kernel memory and mark it as copy-on-write to preserve the correct API semantics. A pointer to the remapped page is added to the socket buffer data structure to keep our changes transparent to the rest of the stack (figure 2). We do not checksum data as this can be offloaded to the hardware but the original

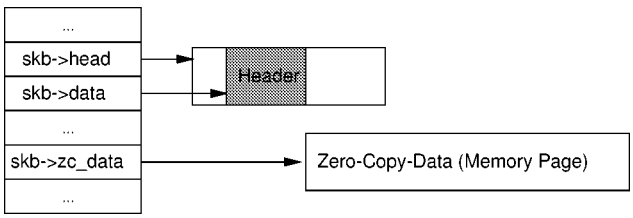

Figure 2. Enhanced socket buffer with pointer to a zero-copy data part (memory page).

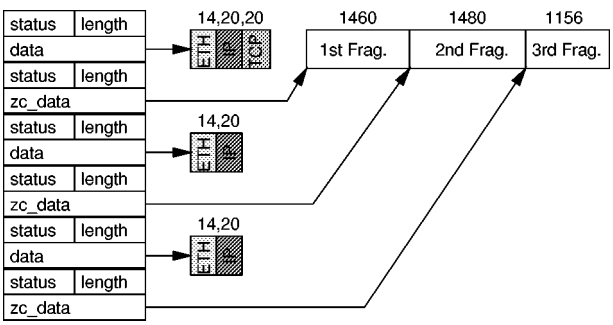

Figure 3. Descriptor list with six entries showing a defragmented 4 KByte packet. The header data consists of an Ethernet and an IP part, in the first packet additionally a TCP part. The numbers indicate the length of the parts in Byte.

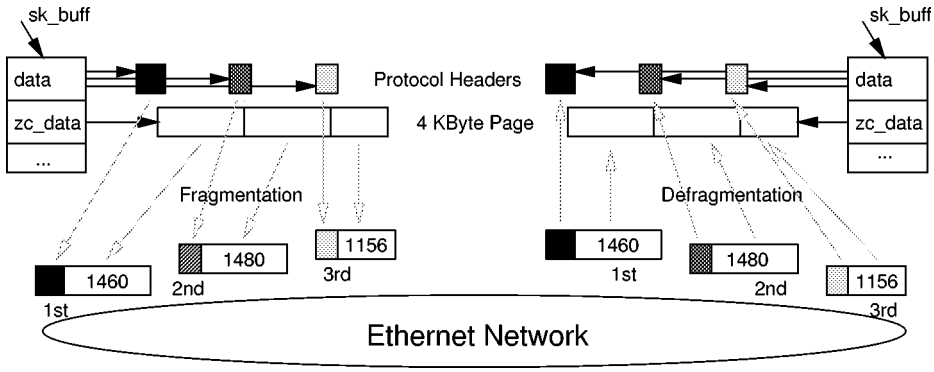

Figure 4. Fragmentation/defragmentation of a 4 KByte memory page is done by DMA through the interface hardware.

TCP/IP stack has not been touched, as the protocol headers are generated and stored in the buffer exactly the same way.

On the receiver side, the original device driver controls the DMA operation of the incoming frame to a previously reserved socket buffer during an interrupt. Later, after the time-critical interrupt, the data is passed to the TCP/IP-layer immediately. When the application reads the data, a copy to the application buffer is made. Our implementation differs from this schema as we reserve an entire memory page from the kernel to store a series of incoming fragments. Once this page holds an entire 4 KByte IP packet, it is processed by the original stack and then remapped to the address space of the receiving application. Remapping instead of copying is possible, if the following conditions are met:

1. The user buffers are page aligned and occupy an integral number of MMU pages.
2. The received messages must be large enough to cover a whole page of data.
3. The `zc_data`-pointer must point to the data of the zero-copy socket buffer.

If one of the conditions is violated our implementation falls back to the normal operation of the Linux TCP/IP stack and hereby preserves the original copy semantics of the traditional socket interface.

3.2.2. Speculative defragmentation

The fragmenting Ethernet driver is enabled to send and receive an entire memory page and further features header separation in hardware. In zero-copy mode the TCP protocol stack automatically generates packets of 4 KByte whenever possible, and the driver decomposes them into three IP-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. This scheme of descriptor management permits to use the DMA-engine of the NIC in order to fragment and defragment frames directly from and into memory pages that are suitable for mapping between user and kernel space.

A descriptor entry comprises a pointer to the data in the buffer, fields for status and the buffer length. Figure 3 shows a snapshot of a descriptor list after the buffers were written by the DMA.

The problem with current network interfaces is that descriptors must be statically pre-loaded in advance and that a proper separation of header and data is only possible by guessing the length of the header.[1] For the zero-copy implementation the length of IP- and TCP-header fields must be pre-negotiated and assumed correctly. If an incoming frame does not match the expected format of an IP fragment or contains unexpected protocol options, the zero-copy mode is aborted and the packet is passed to the regular protocol stack and copied. In the current implementation every unexpected non-burst packet causes zero-copy operation to be disrupted.

When choosing another protocol (e.g., IPv6 or a special purpose protocol) the only thing that has to be altered in the driver are the length values of the header and data fields that are speculated on.

3.2.3. Packet transmission and reception

In place of the standard IP-stack, our NIC device driver is responsible for fragmenting packets of the 4 KByte payload into appropriate Ethernet fragments. To do this, the driver emulates a virtual network interface that pretends to send large Ethernet frames. The fragmentation is done in a standard way by setting the *More_Fragments* flag and the proper offsets in the IP-header as outlined in figure 4. Moving the fragmentation/defragmentation from the IP-stack to the device driver therefore permits to offload this work intensive task to the NIC hardware.

Upon packet arrival, the controller logic transfers the header and the payload into the buffers of the host memory as designated by the speculatively pre-loaded receive descriptor list. After all the fragments are received or, alternatively, after a timeout is reached, an interrupt is triggered and the driver checks whether the payloads of all received fragments have been correctly written to the corresponding buffer space. In the case of success, i.e., if all the fragments have been received correctly, the IP-header is adapted to the

[1] The Hamachi chip does protocol interpretation at runtime to calculate checksums, 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.

new 4 KByte frame and the packet is passed further to the original IP-stack. If the speculation fails or the driver has received an interfering packet between the individual fragments of a 4 KByte frame, some data ends up in a displaced location. As soon as the device driver detects this condition the receiving DMA is stopped and the data has to be copied out of the wrongly assumed final location into a new socket buffer and afterwards passed forward for further processing. 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.

4. A network control architecture to improve successful speculation

The problem with our speculative solution is that multiple concurrent blast transfers to the same receiver will result in interleaved streams, garbling the zero-copy frames and reducing the performance due to frequent miss-speculation about the next packet. 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 a *fast mode*. Unlike in blast facilities, this control architecture does not necessarily involve new protocols that differ from regular IP nor an explicit scheduling of such transfers through a special API.

No precise knowledge of this dedicated network control architecture is required to understand the architectural issues of speculative defragmentation for fast transfers. However the knowledge of our admission control mechanism might be needed for the proper interpretation of our performance results in section 5.

4.1. Admission control for fast transfers

To achieve an exclusive allocation of a host-to-host channel we have successfully implemented a distributed admission control mechanism at the driver level with especially tagged Ethernet packets that are handled directly by Ethernet driver.

A sender requests a zero-copy burst with a *Fast_Req* packet to the receiver which is answered by a *Fast_Ack* or a *Fast_NAck* packet. Although the round trip time of such a request is about 30 μs, only the invocation of the protocol is delayed and not the data transfer. The transfer can be started immediately with low priority and as soon as an acknowledgment arrives, the zero-copy mode can be turned on (in our software implementation this simply means that the packets are sent as three fragmented IP packets with a total of 4096 Bytes, as described in section 3.2.3). In the rejection case *Fast_NAck* the fast transfer is either delayed or throttled to a lower bandwidth in order to reduce interference with the ongoing zero-copy transfer of another sender. We prove in the performance analysis section that such low priority data-streams do not affect the bandwidth of a fast transfer in progress, but that they are essential to guarantee deadlock free operation.

4.2. Implicit versus explicit allocation of fast transfers

Using the protocol described above, it is also possible to hide the entire network control architecture with slow/fast transfers from the user program and to implicitly optimize the end-user communication performance with standard TCP/IP sockets. Fast transfers can be automatically requested and set up so that certain data transfers benefit from a highly increased bandwidth.

As an alternative, the selection of the transfer mechanisms can be put under application control through an API. For this option we implemented an I/O-control (`ioctl`) call which sends requests for fast transfers and returns the answers of the receivers, so that fast transfers can be put under user control.

5. Performance results

5.1. 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 a 1000BaseSX Ethernet permits transfer rates of about 125 MByte/s – at least in theory.

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 can reach 126 MByte/s, as we measured for large burst transfers between a PCI card and the main memory of the PC. The performance for a memory to memory copy by a Pentium II CPU is at 92 MByte/s for the Intel 440 BX chipset operating an SDRAM based memory system clocked with 100 MHz.

As workstation main memory bandwidth has been and is expected to continue increasing more slowly than the point-to-point bandwidth of communication networks, copying between system and application buffers is and will be the major bottleneck in end-to-end communication over high-speed networks.

5.2. Measured best case performance (gains of speculation)

Regular 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 5). Even after OS support for zero-copy (e.g., remapping with COW) is added, the performance is still disappointingly low, because with Ethernet there is still the defragmenting copy. A similarly low performance is achieved with the speculative defragmentation alone (without a zero-copy embedding into an OS). The bandwidth increases from 42 to 45 MByte/s, because our current implementation of the speculative defragmenter attempts to receive three packets in a row and therefore reduces the interrupts. Handling just three packets at the time is considerably less than the huge transfers considered in previous work about blast protocols and so the minimal transfer length for half of peak speed (half-length) is much better.

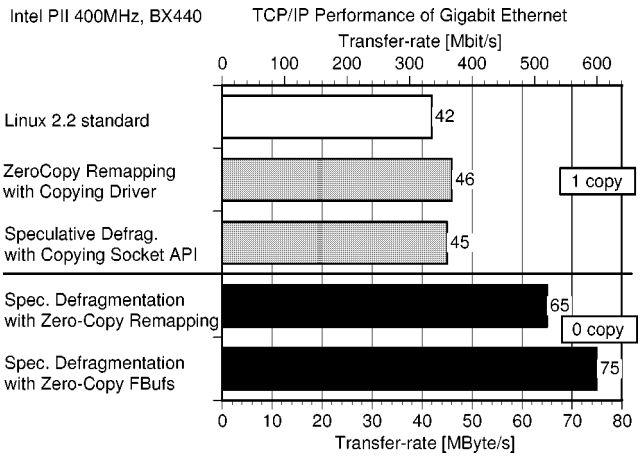

Figure 5. Throughput of large data transfers over Gigabit Ethernet (light grey bar). In combination with a zero-copy interface (FBufs) the throughput is increased from 42 to 75 MByte/s (black bar). If the zero-copy embedding as well as the speculative defragmentation is used alone the performance does not increase much (dark gray bars), since data is still copied in the driver for packet defragmentation or in the upper layer of the system between user and kernel space. 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 and a 32 bit/33 MHz PCI-bus.

To see a leap in the performance we must combine all zero-copy technologies to eliminate the last copy. After the integration of our defragmenting driver into the zero-copy OS environment, the performance of the TCP/IP stack climbs to 65 MByte/s for fast transfers in a “page remapping” environment and to 75 MByte/s in a “fast buffers” environment. The “page remapping” approach is slightly slower than the “fast buffers” approach since some expensive memory mapping operations are performed during the transfer itself in the first case instead of during startup phase in the second case.

5.3. Performance of fallback (penalties when speculation fails)

A first “backward compatibility” fallback scenario measures the performance of a sender that dumps fragmented 4 KByte TCP packets to an unprepared standard Linux receiver. As mentioned before, we use standardized IP-fragmentation and so every receiving protocol stack is able to handle our fast streams without any problems at normal speed (see figure 6). 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 into the cleanup code at the receiver is required. The maximum cost of this worst case is depicted with a standard sender transmitting to a receiver in zero-copy mode. The overhead of the cleanup code reduces the performance from 42 MByte/s to about 35 MByte/s. If the device driver detects that a packet not

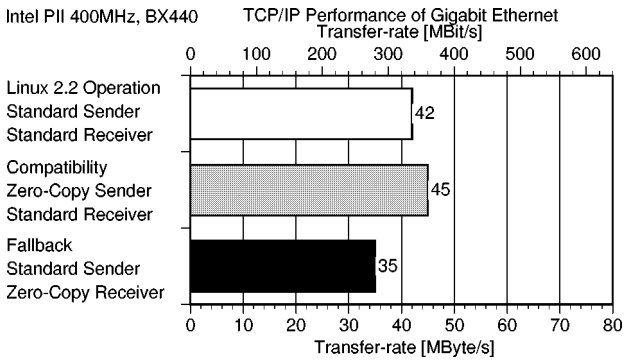

Figure 6. TCP throughput across a Gigabit Ethernet for two fallback scenarios. There is no penalty for handling sender fragmented packets at an unprepared receiver in normal receive mode. Only if a standard sender is interrupting a burst into a zero-copy receiver, cleanup and resynchronization after each packet is needed. This case should be infrequent and remains unoptimized, but it still performs at 35 MByte/s which is not much slower than the standard implementation at 42 MByte/s.

Table 1
Effect of interferences on a transfer with the `ttcp` benchmark sending 100 000 zero-copy packets containing 4096 Byte. The bandwidth is still much better even if the zero-copy transfer is highly interrupted.

Packets interfered	Failed ZC-packets	Bandwidth (MB/s)
0	2	75
100	15	73
10000	28	63

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. This explains the small performance reduction.

For pre-scheduled communication, the cleanup scenario is already highly unlikely. By an appropriate network control architecture that coordinates the senders and receivers in a cluster (see section 4) the probability of such occurrences can be reduced substantially. Table 1 shows the bandwidths achieved with infrequently interrupted zero-copy transfers.

In section 6 we propose a simple packet filter logic for the NICs which would separate any burst streams from all the other traffic. So the miss rate could be further reduced.

5.4. Rates of success in real applications

Table 2 shows some packet-arrival hit/miss statistics of two application traces gathered on our cluster of PCs. The first trace is taken from a distributed Oracle database executing query 7 of a TPC-D workload (distributed across multiple nodes of cluster by an SQL parallelizer) and the second trace is the execution of an OpenMP SOR code using the TreadMarks DSM system (distributed shared memory). In the Oracle case with TPC-D workload, two bursts containing results from queries are simultaneously communicated back

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.

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

Table 3
Execution time improvements for a TreadMarks SOR code running on 3 PCs, communicating with two page sizes over different networks. Just a small performance improvement can be measured as the application is very much latency sensitive. With larger pages the improvement is higher even if more data has to be communicated.

Packet size (KByte)	Number of pages	Execution times (s)		
		Fast Ethernet	Gigabit Ethernet	ZC-Gigabit Ethernet
4	21218	41.4	39.7 (−4.1%)	39.3 (−5.1%)
16	5509	40.0	37.1 (−7.3%)	36.6 (−8.5%)

to the master at the end of the distributed queries. This leads to many interferences, which need to be separated by the proposed control architecture.

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. The structure of the calculations or of the DSM middleware properly serializes and schedules the transfers so that the speculation does work almost perfectly.

5.5. Execution times of application programs

For a further investigation of performance that goes a step beyond small microbenchmarks, we considered measuring standard benchmarks (like the NAS Parallel Benchmark [2]) on top of our new communication infrastructure as well as full speed runs with the applications used for the trace based studies in the previous paragraphs. Unfortunately, we consistently ran into performance bottlenecks within the communication middleware for most meaningful applications. NAS uses MPI, which needs a modification and a major rewrite to support zero-copy communication. The TPC-D data mining benchmark under ORACLE 8.0 results in nice traces and useful hints about traffic patterns, but we cannot get that middleware to generate data at speeds higher than 1.4 MByte/s [24]. With such low communication demands Gigabit Ethernet does not make sense and the execution times compared to Fast Ethernet remain the same. The TreadMarks system can take some advantage of the lower CPU utilization and the higher bandwidth. Table 3 shows an overall improvement of the SOR example used in the last section of 4.1% (7.5% with 16 KByte pages) with Gigabit Ethernet and 5.1% (8.5% with 16 KByte pages) with zero-copy Gigabit Ethernet. The disappointing magnitude of the improvement can be explained by a communication protocol that is highly sensitiveness to latency and so a higher bandwidth does not result in better performance. With larger pages the improvement is a bit better as more data has to be communicated. Without a cleanup of the TreadMarks communication code, the gains of zero-copy communication remain marginal because the middleware requires with many internal data copies. This looks pretty much like a chicken and egg problem as all current middlewares still rely heavily on copying semantics internally. Without a widespread Gigabit communication infrastructure in place, there is no trend in commercial software that makes use of such high speed networks.

As a third application code we investigated the data streaming tool *Dolly* that was designed by our group [22]. Dolly is used to distribute data streams (disk images) to a large number of machines in a cluster of PCs. Instead of network multicast Dolly uses a simple multi-drop chain of TCP connections. Using a logical network topology eliminates the server-bottleneck and renders a data distribution performance that is nearly independent of the number of participating nodes. The tool is used for distributing hard disk partitions to all nodes in widespread clusters, enabling fast operating system installation, application software upgrades, or distributing datasets for later parallel processing. In the different context of [22] we give a detailed performance prediction model for this application on a variety of different hardware and calculate the maximum achievable bandwidth of data distribution by a simple analytical model. The model accounts for the limiting resources inside the nodes and predicts a maximal streaming bandwidth for a given network topology. The prediction is based on the flow of the data streams in the nodes and their usage of the resources. These limiting resources could be either the memory subsystem,

Table 4
Predicted and measured bandwidths for a data distribution over a logical multi-drop chain. As the CPU is the bottleneck, not the network, there is just a performance improvement when the communication copies can be eliminated. This is reflected by the model as well as by the measured performance.

Dolly streaming (MB/s)	Fast Ethernet	Gigabit Ethernet	ZC-Gigabit Ethernet
Modeled	11.1	11.1 (+0%)	14.7 (+32%)
Measured	8.8	9.0 (+2%)	12.2 (+39%)

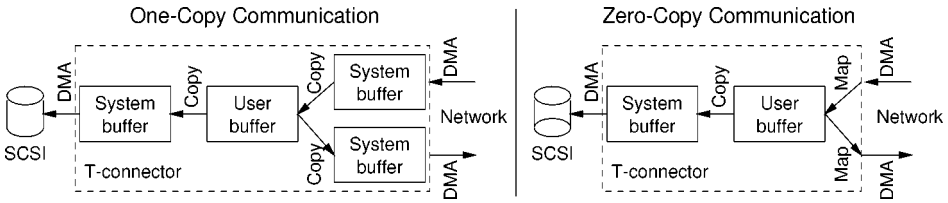

Figure 7. Schematic data flow of an active node running the *Dolly* client.

the I/O bus (which is used for disk and network I/O), the attached hard disk drives or the CPU utilization. While the absolute speeds with our advanced zero-copy communication architecture did not break records, the model explains a performance improvement of nearly 40% very accurately. The limiting factor on current PCs is the memory system utilization and so the fewer copies result in significant performance improvements and with it, in shorter down-times for the software maintenance of our clusters.

For a partition cast over a logical multi-drop chain (see table 4) the model predicts the same bandwidth for Fast Ethernet and Gigabit Ethernet, as it identifies the CPU and memory system of the nodes as the main bottleneck; the network is much faster. If the zero-copy stack is used, i.e., the communication copies are eliminated (see figure 7), the improvement due the reduced load on CPU and memories is reflected by the model as well as by the measured streaming performance. While the transition from Fast Ethernet to Gigabit Ethernet result in a marginal performance gain of 2%, there is a quite an impressive leap in performance of 39% when using Gigabit Ethernet with our new zero-copy stack based on speculative defragmentation. Since the streams of a multi-drop chain are quite static and well controlled, there are close to no mispredictions in the defragmentation process for the application Dolly.

6. Enhanced hardware support for speculative zero-copy

In section 3.2 we showed that, based on speculation techniques, it is possible to implement a 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, e.g., when the defragmenter made wrong guesses about the packet order. Speculation misses automatically raise the question of using prediction hardware to improve the accuracy of the guesses. We therefore propose three simple extensions 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 current off-the-shelf Ethernet hardware.

6.1. Control path between checksumming- and DMA-Logic

Many networking adapters already do protocol detection to help with the check-summing of the payload but this detection information cannot be used to control the transfer of the frames to the host. As a first improvement to today's Ethernet NICs we propose to introduce an additional control-path between the checksumming logic with its protocol-matching engine and the DMA-Logic. This makes the protocol information available to the DMA-Logic and allows to reliably separate the headers from the payload data. In section 3.2 we showed that this does not imply a totally different protocol stack implementation, but for clients using the separation it would greatly improve the rate of success even if the support does not work for all the frames (e.g., for frames with different TCP options).

6.2. 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 a zero-copy implementation the difficult part is to deposit the incoming, possibly fragmented payload directly into its final destination in memory. The previous work with ATM or VIA [11,25] 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 data copies to solve the problem.

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

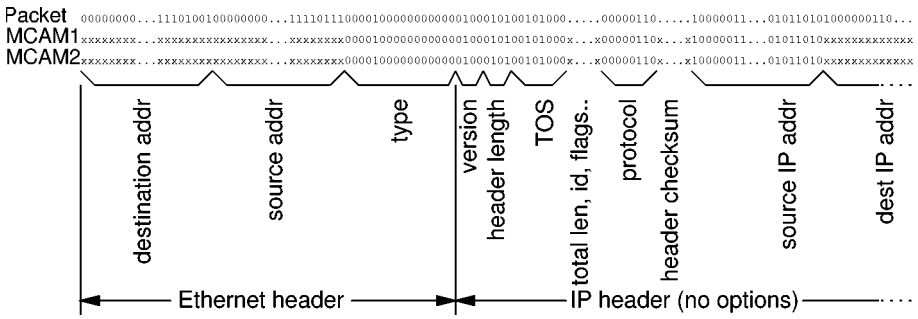

Figure 8. The bit stream of every incoming packet is matched against a content addressable match register (match cam) to detect packets to be handled by the zero-copy defragmenter. Matching against an "x" (do not care) in the match register is implemented with a second mask register. Multiple match register 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.

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.

6.3. Content addressable protocol match registers

The goals of using standard Ethernet in cluster computing is to work with mass market equipment for the interconnects and for the switches. 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 at the Ethernet frame level, e.g., with a modified protocol ID, but this could cause problems with smart switches, so using the *Type-of-Service* bits within the IP header might be more appropriate.

As a third enhancement we propose to implement a stream detection-logic with a simple matching register to separate special zero-copy transfers from ordinary traffic.

As mentioned before, many networking adapters do already support a static protocol detection to help with the checksumming of the payload. But this information is not made available to control the transfer of the frames to the host and the built in detection is 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 then 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 mask-able with a "don't care"-option and it would make sense to provide some modest associativity (e.g., 2 way or 4 way) for matches. An example of a packet match is given in figure 8.

The best location of this protocol matching hardware strongly depends on the VLSI implementation of the Gigabit Ethernet interface, as shown in figure 9. 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.

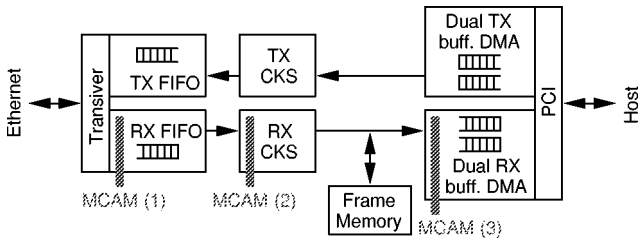

Figure 9. Schematic flow of the packets through a typical Gigabit Ethernet adapter. The gray bars indicate three possible locations (1), (2) or (3) for the proposed protocol match cam mechanism. We recommend the location (1) at the beginning of the RX FIFO since it provides the matching information as early as possible.

Match CAMs were extensively 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 [14] consumed less than 2000 gates. Matching registers for protocol detection are already present in many NICs although they usually control only the checksumming logic and not the descriptor selection for DMA.

7. Conclusions

The small packet size of standard Gigabit Ethernet prevents common zero-copy protocol optimizations unless IP packets can be fragmented most efficiently at the driver level without involving any additional data copies. Accurate fragmen-

tation and defragmentation of packets in hardware remains impossible with most existing Gigabit Ethernet network interface chips unless a *speculative approach* is taken.

With speculation techniques it becomes possible to implement true end-to-end zero-copy TCP/IP based on some simple existing network adapters. A substantial fraction of the peak bandwidth of a Gigabit Ethernet can be achieved for large transfers while preserving the standardized socket API and the TCP/IP functionality. The study of the speculation hit rates within three applications (the SOR with TreadMarks DSM, a TPC-D benchmark with Oracle and our own Dolly tool for partition casting) shows that speculation misses are quite rare and that they can be further reduced in the highly regular Ethernets connecting the compute nodes of a Cluster of PCs.

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 solution for speculation failure. 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 re-mapped to the address space of the communicating application program. All checks whether the incoming packets are handled correctly, i.e., according to the protocol, are deferred until a burst of several packets arrived and only if the speculation indeed misses, the 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 and better speculation about handling incoming packets most efficiently. Two promising approaches are outlined, the first approach suggests a hardware extension of the network interface using a few simple protocol match CAM (content addressable memory) registers that classify incoming packets into two different categories: high speed zero-copy traffic and regular TCP/IP traffic. The match cam registers would control the selection of DMA descriptors used to store the packets in memory from a different descriptor lists. The second approach uses a dedicated protocol for admission control that guarantees exclusive access for one burst data stream.

Our implementation of a speculative Gigabit Ethernet driver with a speculative defragmenter is embedded into an OS setting with well known 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 achieved 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. The benefits of a zero-copy communication architecture are not limited to high peak bandwidth numbers, but also include a much lower CPU utilization that will improve the performance of many application. Our Dolly tool for data streaming and disk cloning over Gigabit Ethernet was measured to perform 39% better with our zero-copy communication architecture than with a standard protocol stack.

Our approach of speculative processing in hardware, along with cleanup in software as a fallback, seems to enable a set of simple new solutions to overcome some old performance bottlenecks in network interfaces for high speed networking.

References

[1] Alteon WebSystems Inc., Jumbo frames whitepaper, Product Webpage: `www.alteonwebsystems.com`.

[2] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga, NAS Parallel Benchmark, Technical Report RNR-94-007, NASA Ames Research Center, March 1994.

[3] N.J. Boden, R.E. Felderman, A.E. Kulawik, Ch.L. Seitz, J.N. Seizovic and W. Su, Myrinet – A Gigabit per second local area network, IEEE Micro 15(1) (February 1995) 29–36.

[4] 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)*, Seattle, WA, October 1996 (USENIX) pp. 277–291.

[5] J.C. Brustoloni and P. Steenkiste, Copy Emulation in Checksummed, Multiple-Packet Communication, in: *Proc. INFOCOM’97*, Kobe, Japan, April 1997 (IEEE).

[6] J.B. Carter and W. Zwaenepoel, Optimistic Implementation of Bulk Data Transfer Protocols, in: *Proc. of the 1989 Sigmetrics Conference*, May 1989, pp. 61–69.

[7] H.K. Jerry Chu, Zero-Copy TCP in Solaris, in: *Proc. of the USENIX 1996 Annual Technical Conference*, San Diego, CA, USA, January 1996 (The USENIX Association) pp. 253–264.

[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*, Asheville, NC, December 1993, pp. 189–202.

[10] C. Dubnicki, E.W. Felten, L. Iftode and K. Li, Software support for virtual memory-mapped communication, in: *Proc. 10th IEEE Int. Parallel Proc. Symp.*, Honolulu, HI, April 1996, pp. 372–381.

[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) (March–April 1998) 66–76.

[12] P. Geoffray, L. Prylli and B. Tourancheau, BIP-SMP: High performance message passing over a cluster of commodity SMPs, in: *Proc. SC99*, Portland USA, November 1999 (ACM).

[13] GigaNet Inc., Product Webpage: www.giganet.com.

[14] T. Gross and D. O’Hallaron, *iWarp: Anatomy of a Parallel Computing System* (MIT Press, 1998).

[15] InterProphet, SiliconTCP^{TM}, Product Webpage: `www.interprophet.com`.

[16] 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*, Albuquerque, NM, November 1991 (IEEE) pp. 664–673.

[17] Ch. Kurmann and T. Stricker, A comparison of three Gigabit technologies: SCI, Myrinet and SGI/Cray T3D, in: *SCI Based Cluster Computing*, eds. H. Hellwagner and A. Reinefeld (Springer, Berlin, 1999). An earlier version appeared in Proc. of the SCI Europe’98 Conference, EMM-SEC’98, 28–30 September 1998, Bordeaux, France.

[18] F.W. Miller, P. Keleher and S.K. Tripathi, General Data Streaming, in: *Proc. 19th IEEE Real-Time Systems Symposium*, Madrid, December 1998 (IEEE) pp. 232–241.

[19] F. O'Carroll, H. Tezuka, A. Hori and Y. Ishikawa, The design and implementation of zero copy MPI using commodity hardware with a high performance network, in: *ICS '98. Conference Proceedings of the 1998 International Conference on Supercomputing*, Melbourne, July 1998 (ACM) pp. 243–250.
[20] S.W. O'Malley, M.B. Abbot, N.C. Hutchinson and L.L. Peterson, A transparent blast facility, Internetworking: Research and Experience 1(2) (December 1990).
[21] V.S. Pai, P. Druschel and W. Zwaenepoel, I/O-Lite: A unified I/O buffering and caching system, in: *Proc. of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)* (1999) pp. 15–28.
[22] F. Rauch, Ch. Kurmann and T.M. Stricker, Partition cast – modelling and optimizing the distribution of large data sets on PC clusters, in: *Euro-Par 2000 Parallel Processing, 6th International Euro-Par Conference Munich*, Munich, 2000, Lecture Notes in Computer Science, Vol. 1900, eds. A. Bode, T. Ludwig, W. Karl and R. Wismüller (Springer). Also available as Technical Report 343, Department of Computer Science, ETH Zürich, `http://www.inf.ethz.ch/`.
[23] R. Seifert, *Gigabit Ethernet: Technology and Applications for High-Speed LANs* (Addison-Wesley, 1998). ISBN: 0201185539.
[24] M. Taufer, Personal communication, ETH Zurich, September 2000.
[25] 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, December 1995 (ACM).

Christian Kurmann received his bachelors and masters degrees in computer science engineering from the Swiss Federal Institute of Technology (ETH) Zürich, Switzerland, in 1996. He is currently a doctoral student at ETH in Zürich, Switzerland, and is working on high performance computing with CORBA and networking for clusters of PCs. Christian Kurmann is a student member of the ACM SIGCOMM and the IEEE Computer Society.

Felix Rauch received his bachelors and masters degrees in computer science engineering from the Swiss Federal Institute of Technology (ETH) Zürich, Switzerland, in 1997. He is currently a doctoral student at ETH in Zürich, Switzerland, and is working on distributed high performance file systems for clusters of PCs. Felix Rauch is a student member of the ACM and the IEEE Computer Society.

Thomas M. Stricker is an assistant professor of computer science at the Swiss Federal Institute of Technology (ETH) in Zürich. His interests are in architectures and applications of Clusters of PCs that are connected with a gigabit network. Thomas Stricker received his Ph.D. from Carnegie Mellon University in Pittsburgh, USA, in 1996. He participated in several large systems building projects including the construction of the iWarp parallel machines. He is a member of the ACM SIGARCH, SIGCOMM and the IEEE Computer Society.