Speculative Defragmentation — A Technique to Improve the Communication Software Efficiency for Gigabit Ethernet

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

Abstract

Modern massively parallel computers are built from commodity processors and a trend towards commodity interconnect components for Clusters of PCs (CoPs) is visible. Most of today's networking solutions are still proprietary, but they do connect to standard buses (i.e. PCI) and in the near future the networking solutions of the Internet (e.g. Gigabit Ethernet) can offer Gigabit speeds at mass market prices.

Cluster platforms like CoPs offer good compute performance, but still they cannot yet utilize the potential of Gigabit/s communication technology, at least not with commodity network adapters like Ethernet NICs and standard protocols like TCP/IP. While the speed of Ethernet has grown to 1 Gbit/s the functionality and the architectural support in the network interfaces remained the same for more than a decade, so that the memory system becomes a limiting factor. To sustain the raw network speed in applications, a "zero-copy" network interface architecture would be required, but for all widely used stacks a last copy is required for the (de)fragmentation of the transferred network packets, since Ethernet packets are smaller than a page size.

Correctly defragmenting packets of various communication protocols in hardware is an extremely complex task. We therefore consider a speculative defragmentation technique, that can eliminate the last defragmenting copy operation in zero-copy TCP/IP stacks on existing hardware. 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. For an evaluation of our ideas we integrated a network interface driver with speculative defragmentation into an existing protocol stack and added well known page remapping and fast buffers strategies. Measurements indicate, that we can improve the performance for Gigabit Ethernet over a standard Linux 2.2 TCP/IP stack by a factor of 1.5–2 for uninterrupted burst transfers. Furthermore, our study demonstrates good speculation success rates for a database and a scientific application code on a cluster of PCs.

1. Introduction

High 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.

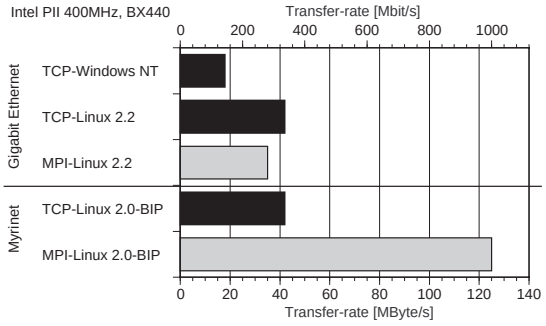

Figure 1. Throughput of large data transfers over Gigabit/s point-to-point links with standard and special purpose networking technology and communication system software.

Three prominent examples of current Gigabit/s networking products are Gigabit Ethernet [18], Myrinet [2] and the Scalable Coherent Interconnect (SCI) [14]. 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 among the three examples. Disappointing communication performance results if standard interfaces and protocols such as the socket API and TCP/IP are used. Figure 1 shows the data rates achieved for large transfers with the French MPI library BIP [10] and with standard TCP/IP proto-

col stacks 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 even with commodity platforms using a PCI bus based network interface card. The TCP performance (black bars) is about one third of the maximal achievable rate and shows the problem we are focusing on, the poor performance with a TCP standard interface and Gigabit Ethernet. One reason of the good performance of MPI over Myrinet are large packet sizes and that there is no need for packet fragmentation.

Gigabit Ethernet, like all previous versions of Ethernet, has been designed for an unacknowledged, connection-less delivery service and provides point-to-point connections supporting bi-directional communication including link-level (but no end-to-end) flow control. One of the big advantages of Gigabit Ethernet is the backward compatibility with its predecessor, so called Fast Ethernet (100 Mbit/s). The downside is that its maximum transmission unit (MTU) of Ethernet (1500 Byte) still remains smaller than the page size of any processor architecture. Ethernet driver systems thus use at least one data copy to separate headers from data, that renders efficient zero-copy techniques useless, unless defragmentation is done in hardware. The current standards therefore prevent efficient zero-copy techniques from being used instead of simple copy-techniques.

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 PCI based network interface cards (NIC). Looking more closely at the currently available hardware solutions 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 so that under certain assumptions, a rigorous true zero-copy protocol might still be feasible.

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. The paper is structured as follows: After an overview of related work in zero-copy software and hardware architectures in Section 2 we describe the speculation technique in Section 3 that provides the qualitative and the quantitative foundation for our improved Gigabit Ethernet network architecture targeted towards compute clusters. In Section 4 we suggest ways to improve the likelihood for a successful speculative transfer with a dedicated network control architecture for cluster computing. Finally, in Section 5, we present some highly encouraging performance results and conclude in Section 6.

2. Related Work in the Area of Zero-Copy Communication

Previous work resulting from the early projects in network computing widely acknowledged the observation 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/s speeds [13].

Since today's high speed networks, I/O systems and hierarchical memory systems operate at comparable bandwidth of about 100 MByte/s, one of the most important challenges for communication system software is therefore 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 [5].

1. **User-Level Network Interface (U-Net) or Virtual Interface Architecture (VIA)**: Low level hardware abstractions for the network interfaces suggest to leave the implementation of the communication system software to libraries in user space [19, 9, 8].

2. **User/Kernel Shared Memory (FBufs, IO-Lite)**: The technique relies on shared memory semantics between the user and kernel address space and permits to use DMAs for moving data between the shared memory and network interface. Such drivers can also be built with per-process buffer pools that are pre-mapped in both the user and kernel address spaces [7, 17].

3. **User/Kernel Page Remapping with Copy on Write**: 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 [5].

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]. The basic 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 more than a decade ago. These investigations were in the context of:

4. **Blast transfer facilities**: Blasts are special protocol for large transfers, 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, and e.g. in [4], the data parts are directly written to user space, while in [16] the pages with the contiguous data parts are remapped from kernel to user space.

In the ten years since these investigations on blast transfers, the network and memory bandwidths have increased by two orders of a magnitude and architectural limitations apply. However, the architecture of popular network interfaces for Fast and Gigabit Ethernet has hardly changed. Blast transfers use dedicated protocols and none of the blast techniques have made their way into production clusters. Our work is different, since it aims at performance improvements by improving the integration of fast transfers with existing hardware and protocol stacks and, as a novelty, we add the distinct viewpoint of speculative protocol processing with cleanup code upon failure.

2.2. Gigabit Networking Alternatives and their Solution for Zero-Copy

For System Area Networks (SAN) of the clusters of PC platform a few highly attractive SAN alternatives to Gigabit Ethernet are available, those comprise Myrinet [2], SCI [6], Giganet [11] and ATM-OC12. There are some relevant architectural differences between these interconnect technologies and Gigabit Ethernet. In most special purpose 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 fundamental for zero-copy strategies and efficient driver software.

[1]There are many successful prototypes, but most of them include semantic restrictions for zero-copy buffer management and only work with network technologies supporting large frames. A good overview is given in [3]. A better operating system support for I/O streams is described in [15].

Myrinet interconnects support 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 provides significant processing power through a user programmable RISC core with large staging memory. Under these circumstances there is much less justification for a TCP/IP protocol stack. Similarly the bulk data transfer capability of SCI interconnects relies on hardware for 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 (MTU of 53 Byte). However, with such fast links and such small packets, there is no hope for defragmentation in software and ATM adapters must provide this functionality entirely in hardware. The rich ATM functionality comes at a hardware cost, that might be too high for most PC clusters. A similarly expensive hardware solution is SiliconTCP^{TM} [12]. It implements a TCP/IP stack in hardware which leads to the benefit of very low main processor utilization but in current implementations it cannot keep up with a Gigabit/s rate of fast interconnects.

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. This makes the software defragmentation into whole page sizes needless but nevertheless, the concept of Jumbo Frames does not solve the problem of header/payload separation hereby contradicting the idea of using mainstream networking technologies in a cluster 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 packet switched networks, the presence of Jumbo Frames will result into high latency for small packets traveling in the same network.

Instead of demanding ever increasing packet sizes we suggest to incorporate a modest additional hardware support for defragmentation of small Ethernet packets (similar to ATM cells) which results in lower overall latencies and high bandwidth transfers for highly fragmented packets. As such interfaces are still not available today, we enlist techniques of specula-

tion to go the route of efficient defragmentation at the driver level in software as this is already properly defined by the IP over Ethernet standard. Due to speculation this can be done with the existing network interface hardware.

3. Enabling Zero-Copy for Existing Simple Ethernet Hardware

In order to achieve networking performance between 75 and 100 MByte/s with a Gigabit/s 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 page size) and the usual hardware support of a descriptor based Ethernet network interface, it seems to remain impossible to solve the problem of a true zero-copy transfer because 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 receiver 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 the best case extremely fast at the price of a potentially more expensive cleanup operation if something went wrong. We will show with application statistics that the best case is indeed the 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 data is passed to a 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 is also the case since large transfers are non interruptible. Although, for now, the zero-copy transfers must be scheduled explicitly or implicitly between two nodes (an implicit solution given in Section 4) it can always coexist with normal networking traffic while running without a performance penalty.

3.1. Gigabit Ethernet and its NIC

In our experimental cluster of PCs we use off-the-shelf 400 MHz Pentium II PCs, running Linux 2.2, connected via Gigabit Ethernet by fiber optic cables. Our Gigabit Ethernet test bed comprises a SmartSwitch 8600 manufactured by Cabletron and GNIC-II Gigabit Ethernet interface cards manufactured by Packet Engines. The 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. Advanced interrupt coalescing techniques reduce the number of host interrupts.

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” [5] or as an alternative the “fast buffer” concept [7]. The two different zero-copy techniques demonstrate that the speculative defragmentation technique is an independent achievement and that it works with different OS embeddings.

3.2.1. Speculative Defragmentation

Our fragmenting Ethernet driver manages to send and receive an entire memory page and further features header separation in hardware. The TCP protocol stack therefore automatically generates zero-copy 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. With an *End_Of_Packet* flag in the descriptor status field the controller allows for the bytes of an Ethernet frame to extend across several buffers.

Thanks to this indicator it becomes possible to automatically separate the headers from the payload.

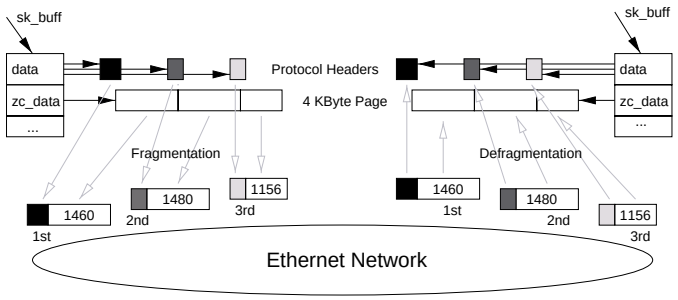

Figure 2. Fragmentation/Defragmentation of a 4 KByte memory page is done by DMA through the interface hardware.

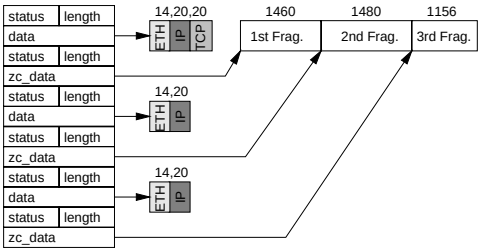

Figure 3. Descriptor list with 6 entries showing a defragmented 4 KByte packet. The header data consists of an Ethernet and an IP part, in the first packet additionally a TCP part. The numbers indicate the length of the parts in Byte.

The problem 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[2]. 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 protocols) 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.

3.2.2. Packet Transmission and Reception

In place of the standard IP-stack, our NIC driver takes on the responsibility of fragmenting packets of 4 KByte payload into appropriate Ethernet fragments. The driver thereby emulates a virtual network interface that allows to send larger 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 2. That means that moving the fragmentation/defragmentation from the IP-stack to the device driver allows to offload the time consuming task to the NIC hardware.

Upon packet arrival, the controller logic transfers the header and the payload into the buffers in host memory 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 success case, if all the fragments have been received correctly, the IP-header is adapted to the new 4 KByte frame and the packet is passed further to the IP-stack. In the failure case, if the driver has received an interfering packet between the individual fragments of a 4 KByte frame, some data ends up in a displaced location because of a mis-speculation. That data has to be copied out of the wrongly assumed final location into a new socket buffer and afterwards passed forward for further processing.

4 A Network Control Architecture to Improve Successful Speculation

The problem with a speculative solution are multiple concurrent blast transfers to the same receiver which 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 so called fast mode. Unlike in blast facilities, this control architecture does not necessarily

[2]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.

involve new protocols, that differ from regular IP or 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 might be beneficial for the 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 specially 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 specially fragmented IP packets of 4096 Bytes in total, as described in Section 3.2.2). In the rejection case of a *Fast_NAck* the fast transfer is either delayed or continued throttled to a low bandwidth in order to reduce interference with the ongoing zero-copy transfer of another sender. We can show in the performance analysis that such low priority data-streams do not affect the bandwidth of a fast transfer in progress but they remain 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 and returns the answers of the receivers, so that fast transfers can be put under user control.

5. Performance Results

5.1. Measured Best Case Performance with Zero-Copy

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 4). After an integration of our defragmenting driver into the zero-copy OS environment, the performance of the TCP/IP stack is increased to 65 MByte/s for fast 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 expensive memory mapping operations are performed during the transfer in the first case and during startup in the second case.

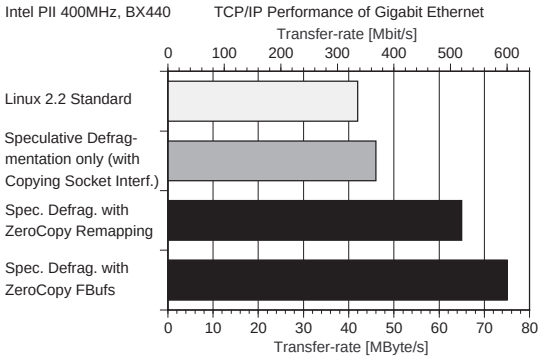

Figure 4. Throughput of large data transfers over Gigabit Ethernet (light grey bar). In combination with a zero-copy interface (FBufs) the throughput is increased from 42 MByte/s to 75 MByte/s (black bar). The speculative defragmentation alone does not increase the performance much (dark gray bar), since 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 either, 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).

5.2. 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 such a stream without any problems at normal speed (see Figure 5).

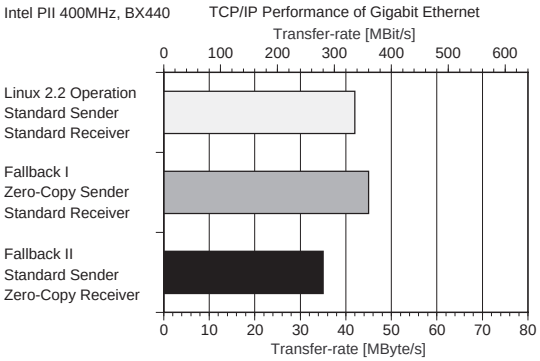

Figure 5. TCP throughput across a Gigabit Ethernet for two fallback scenarios. There is no penalty for handling sender fragmented packets at an unprepared receiver in normal receive mode. Only if a standard sender is interrupting a burst into a zero-copy receiver, cleanup and resynchronization after 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.

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` 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.

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.

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.

5.3. Rates of Success in Real Applications

Table 2 shows the packet-arrival traces of two applications running on our cluster. The first trace is taken from a distributed Oracle database executing query 7 of a TPC-D workload and the second 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 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 the middleware properly serializes and schedules the transfers so that the speculation does work perfectly.

6. 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 fragmentation and defragmentation of packets in hardware remains impossible with most existing Gigabit Ethernet network 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.

Our implementation of a speculative Gigabit Ethernet driver with a speculative defragmenter is embedded in 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.

We conclude that with speculation techniques it is possible to implement true end-to-end zero-copy TCP/IP even with 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 in two applications (SPLASH like FFT and a

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
Zero-copy 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.

Oracle TPC-D benchmark) shows that misses are quite rare and that they can be further reduced by clever network control architectures.

The novel view of speculative hardware support for protocol processing in network interfaces chips could potentially lead to simple, but highly efficient hardware solutions for network adapters that deliver the true Gigabit/s speeds to many important applications on clusters of PCs.

References

[1] Alteon WebSystems Inc. Jumbo frames. www.al teonwebsystems.com/products/white_papers/jumbo.

[2] N. J. Boden, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. 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] J. B. Carter and W. Zwaenepoel. Optimistic Implementation of Bulk Data Transfer Protocols. In *Proceedings of the 1989 Sigmetrics Conference*, pages 61–69, May 1989.

[5] H. K. J. 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.

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

[7] 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.

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

[9] 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.

[10] 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.

[11] GigaNet Inc. Product Webpage: www.giganet.com.

[12] InterProphet. SiliconTCP^{TM}. Product Webpage: www.interprophet.com.

[13] H. 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.

[14] C. 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.

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

[16] S. W. O'Malley, M. B. Abbot, N. C. Hutchinson, and L. L. Peterson. A Transparent Blast Facility. *Internetworking: Research and Experience*, 1(2), Dec. 1990.

[17] V. S. Pai, P. Druschel, and W. 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.

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

[19] 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.