Evaluation of a Zero-Copy Protocol Implementation

Karl-André Skevik, Thomas Plagemann, Vera Goebel
Department of Informatics, University of Oslo
P.O. Box 1080, Blindern, N-0316 OSLO, Norway
{karlas, plageman, goebel}@ifi.uio.no

Pål Halvorsen
UniK, University of Oslo
P.O. Box 70, N-2027 KJELLER, Norway
paalh@unik.no

Abstract

Internet services like the world-wide web and multimedia applications like News- and Video-on-Demand have become very popular over the last years. Since a high and rapidly increasing number of users retrieve multimedia data with high data rates, the data servers can represent a severe bottleneck. Traditional time and resource consuming operations, like memory copy operations, limit the number of concurrent streams that can be transmitted from the server, because of two reasons: (1) memory space is wasted holding identical data copies in different address spaces; and (2) a lot of CPU resources are used on copy operations. To avoid this bottleneck and make memory and CPU resources available for other tasks, i.e., more concurrent clients, we have implemented a zero-copy data path through the communication protocols to support high-speed network communication, based on UVM[6]. In this paper, we describe the implementation and evaluation of the zero-copy protocol mechanism, and we show the potential for substantial performance improvement when moving data through the communication system without any copy operations.

1. Introduction

There has been a tremendous growth in the use of multimedia Internet services, and in particular, applications like News-on-Demand (NoD) and Video-on-Demand (VoD) have become very popular. Thus, the number of users, as well as the amount of data each user downloads from servers on the Internet, is rapidly increasing. Today, contemporary mid-price personal computers are capable of handling the load that such multimedia applications impose on the client system, but in Media-on-Demand (MoD) servers, the potentially (very) high number of concurrent users retrieving data represent a problem. In MoD servers in general, commodity operating systems represent the major performance bottleneck, because *operating systems are not getting faster as fast as hardware* [11]. There are two basic orthogonal approaches for this problem: (1) development of an architecture for a single server that makes optimal use of a given set of resources, i.e., maximize the number of concurrent clients a single server can support; and (2) combination of multiple single servers, e.g., in a server farm or cluster, to scale up the number of concurrent users.

We have concentrated on the first approach. To support multiple concurrent users each retrieving a high data rate multimedia stream, the operating system and server architecture must be improved and optimized. Crucial issues include copy operations and multiple copies of the same data in main memory [12]. A major bottleneck in high throughput systems is the *send()* system call (and equivalents) which copies data from the application buffer in user space to the kernel memory region. This is expensive for several reasons [6]:

- The bandwidth of the main memory is limited, and every copy operation is effected by this.
- A lot of CPU cycles are consumed for every copy operation. Often, the CPU must move the data word-by-word from the source buffer to the destination, i.e., all the data flows through the CPU. This means that the CPU is unavailable during the copy operation.
- Data copy operations affect the cache. Since the CPU accesses main memory through the cache, useful information resident in the cache before the copy operation is flushed out.

To avoid the memory copy bottleneck several solutions have been proposed (for a state-of-the-art overview, see [12]), using mechanisms like programmed I/O, page remapping, shared memory, etc. For example, *Afterburner* [7] and *Medusa* [3] copy data directly onto the on-board memory, using programmed I/O with integrated checksum and data length calculation. Using DMA and a user-level implementation of the communication software, the *application device channel* [8] gives restricted but direct access to an

ATM network adapter; removing the OS kernel from the critical network send/receive path. Additionally, several general cross-domain data copy avoidance architectures are suggested. Already in 1972, *Tenex* [4] used virtual copying, i.e., several pointers in virtual memory referring to one physical page. The *V distributed system* [5] and the *DASH IPC mechanism* [13] use page remapping, and the *container shipping* facility [2] uses virtual inter-domain data transfers where all in-memory copying is removed. Furthermore, *fast buffers* (fbufs) [8] is a facility for I/O buffer management and data transfers across protection domain boundaries, primarily designed for handling network streams using shared virtual memory, combined with virtual page remapping. In *Trapeze*[9], the *uiomove()* function which copies data to and from the kernel is modified to perform page remapping. The *Synthesis*[10] OS kernel generates code for optimized data-movement via the CPU registers.

To experimentally analyze the benefits of zero-copy protocol implementations for gigabit networks, like Gigabit Ethernet or Gigabit ATM, we have implemented a zero-copy data path through the communication protocols to support high-speed communication. It is based on the *UVM virtual memory system* [6] which facilitates virtual memory data movement using new techniques like *page loanout*, *page transfer*, and *map entry passing*. Our purpose is to determine if UVM is a suitable basis for a data transmission mechanism in a media server.

This paper focuses on the design, implementation, and evaluation of our zero-copy mechanism, realized in OpenBSD. Our performance measurements show the potential for substantial performance improvement when moving data through the communication system without any copy operations.

The rest of this paper is organized as follows: Section 2 describes the implementation. In Section 3, we present the results from our performance experiments. Finally, we summarize and conclude the paper in Section 4.

2. Design and Implementation in OpenBSD

The primary goal of this work is to determine the impact that limited memory bandwidth has on network throughput and the potential for improvement by using a copy elimination mechanism.

The protocols UDP/IP and TCP/IP are widely implemented and used. UDP is best suited for experiments to analyze the influence of copy operations to I/O overhead, because TCP attempts, among many other things, to avoid overloading the network. UDP merely transmits data received from an application onto the network as fast as possible. This will likely make the OS overhead more visible with UDP, since TCP performance is also influenced by protocol parameters, the state of the receiver, and the network between the endpoints.

2.1. Data movement

The transfer of data between the application and the kernel is normally performed by copying the data. The major data processing task involved in UDP processing is the checksum calculation, which can be performed in hardware with some NICs (Network Interface Cards). Sending fragmented packets can be done by making a new copy of each fragment and adding headers to each fragment. These copy operations can be avoided if the NIC supports *scatter/gather* DMA, which lets the headers be constructed separately while the buffer is kept in one piece. This makes the implementation at the sender side relatively simple, since application data can in many cases be used without changing their alignment.

Ensuring that the payload of received packets are aligned to application buffers without alignment requirements is more difficult since packets are usually copied from the NIC before being processed. Demultiplexing packets in the NIC might be possible with some cards, but is complicated by fragmentation which can produce situations where the recipient cannot be determined before an entire packet has been reassembled.

Based on this, we will focus on outgoing UDP traffic in the measurements, since this will make it easier to see the impact data copies have on the overhead from the OS. Using the VM system to move data is also simpler with outgoing UDP traffic, compared to incoming UDP traffic. Performance in other situations than those tested might be lower, but the measurements should give an indication of what the upper limit is. Another important reason to focus this work on the data movement at the sender is that we are interested in improving the design of MoD servers to support more concurrent users.

2.2. Platform

Two Intel Pentium II PCs running OpenBSD are used for testing. OpenBSD/i386 supports both the original Mach derived VM system found in 4.4BSD, and UVM [6]. UVM allows the transfer, loan and sharing of VM pages between one or more applications, and between an application and the kernel. The VM page size is 4096 octets on Intel machines, with 4 GB virtual memory addressable through a 32 bits address space.

For our experiments we used two *3com 3c985* Gigabit Ethernet cards based on the *Tigon II* chip [1]. The Tigon chip supports both a 32 and 64 bit PCI data path; the maximum bus bandwidth is 132 MB/s and 264 MB/s, respectively. The NIC can be used in both, 32 bit and 64 bit PCI slots, but the test machines are only equipped with 32 bit slots.

A few simple tests with this equipment showed that the maximum throughput we could reach is significantly lower than the theoretical maximum speed for Gigabit Ethernet. We measured the TCP throughput with *netperf*, a network performance tool, and found it to be 200.84 Mbit/s with a buffer size of 4096 octets. The UDP throughput is 391 Mbit/s for the same buffer size with UDP checksums enabled. This is greater than the maximum value for 100 Mbit/s Ethernet, but clearly below the maximum for Gigabit Ethernet. It is intuitively clear that the performance of the Intel Pentium II PC is not sufficient to fully utilize the available bandwidth of Gigabit Ethernet. It is important to note that this is for the purposes of this study an advantage. Improvements in the protocol implementation that result in reduced data movement and CPU usage should produce a higher throughput and be easy to measure.

2.3. UDP send data path

Sending data on a socket with the *send()* system call results in the *sendit()* function being called in the kernel. This function checks the arguments and then calls *sosend()*. The *sosend()* function performs additional error checking, mainly to ensure that there is room for the data. If the call is non-blocking, as much data as possible is sent. The protocol dependent handling function is then called. For UDP/IP, this is *udp_usrreq*, which calls the output function *udp_output()*. The UDP checksum is calculated before *ip_output()* is called. The IP processing function finds the outgoing interface, calculates the IP header checksum, and calls the interface output routine.

2.4. UVM loan related modifications

The *page loanout* mechanism in UVM allows a *loan* of one or more VM pages between different parts of the OS. The loaned pages are marked COW (Copy On Write). UVM can be used to loan a page belonging to an application to the kernel. This can be used to make the contents of a buffer, which is about to be transmitted over the network, available to the kernel without making a copy of the data. The pages that contain the buffer can then be accessed by the networking subsystem directly.

The UVM loan mechanism is supposed to be controlled by applications through a system call [6], but this interface appears to be unavailable in the public implementation. Therefore, we have modified the kernel to use the loan mechanism without intervention from the application. The decision whether the loan should be performed or a data copy is used, is determined in our implementation by the two following factors:

- *Alignment*. The buffer must be aligned to a page boundary.
- *Range*. The buffer must be continuous; i.e., not composed of multiple buffers at different locations. This is a limitation to simplify the modified code.

In case the loan mechanism is used, the first operation is to perform a loan from the VM map of the application. The next steps are to establish a new mapping in the *mbuf* kernel memory submap and to insert a mapping to the loaned page. Finally, an *mbuf* is updated with an external reference to the data in the loaned page.

We keep the page in the cluster area to allow the usage of the cluster reference count mechanism in the kernel. The mapping is removed and the loan terminated when the kernel is finished with the page. Furthermore, we keep a separate array that has a reference to each page. The entries are needed to properly terminate the loan when the packet has been transmitted.

The *sysctl()* interface is used to activate the loan mechanism and keep track of the number of data transmit requests where the loan mechanism is used.

3. Measurements

In this section, we describe the tests we performed to measure the throughput of the different data movement mechanisms and the results of these tests. The communication endpoints in the tests are identical PCs with 350 MHz Intel Pentium II chips and 64 MB memory.

3.1. Measurement tools

Collecting data from the kernel requires an interface for applications to retrieve the data. We have modified the kernel to allow a software probe to register information about the memory management operations. This is done by using a function which stores a timestamp, a location value, and a third value which can be used for additional information as needed. The code in the probe is kept as simple as possible to avoid influencing the results too much.

To fill the capacity of a Gigabit Ethernet with UDP traffic, it is required to send $1\ Gbps = 1,073,741,824$ bits per second, or $134,217,728$ octets per second. The maximum size of a UDP packet including headers is 65536 octets, but the standard MTU of an Ethernet is 1500 octets. This means that 89478 packets of 1500 octets must be sent each second to fully utilize the network. A time value which is updated at twice that rate, i.e., every $5.59ms$, would be needed to register the maximum time between two successive packets. When taking into consideration that packet sizes might be smaller than 1500 octets, it is desirable to have a certain margin for error. Since it is also useful to measure events shorter than the entire maximum time available for sending a packet, the probe should be able to make measurements with at least microsecond accuracy.

Our probe uses the PCC (Processor Cycle Count) value which is updated every 2.85 ns. This value is placed into a preallocated buffer. The retrieval of this value was measured to take an average of 91 ns, with insignificant variation. In other words, the time resolution of our measurements is good enough and the probe overhead is not significant.

This approach is simple, but can only be used when small amounts of data are stored in the kernel. In order to keep the amount of data small, we used short measurement cycles. After each measurement cycle, we extracted the measurement data from the kernel through the *sysctl()* interface.

3.2. Data movement overhead

In this subsection, we analyze the overhead of moving data from the application, in user space, to the kernel. The test application transmits data with the *send()* system call. The total time for the system call and the part of that time spent copying data from the application to the kernel is measured. It is compared with the results from the same application running on a system which uses the UVM page loan mechanism to make the data available to the kernel.

The total time it takes to transmit a packet (t_{tot}) is in this case the sum of the time needed to copy data from the application to the kernel (t_1), and the time it takes to perform the rest of the operations related to the transfer (t_2). The measured values are t_1, which is recorded from inside the kernel, and t_{tot} which is measured by the application.

The transfer rate (R) from the application is a function of the transfer size (s) and the total time as given in (1).

$$R = \frac{s}{t_{tot}} = \frac{s}{t_1 + t_2} \quad (1)$$

The time it takes to transfer a packet varies with the packet size. The theoretical maximum transfer rate for a given t_2 and s, assuming the data copy can be eliminated entirely, occurs when $\lim_{t1 \to 0} t_{tot}$, as given in (2).

$$R_{max} = \lim_{t_1 \to 0} \frac{s}{t_1 + t_2} = \frac{s}{t_2} \quad (2)$$

Figure 1 attempts to show the maximum potential for improvements by eliminating copy operations, in relation to the maximum speed for a Gigabit Ethernet. It is based on the duration of a given operation on a machine without significant load.

The *Copy only* value in the plot is obtained by assuming that the total output processing time is equal to the time it takes to copy the contents of a data buffer of the indicated size. Based on the *Copy only* plot, it is evident that for the

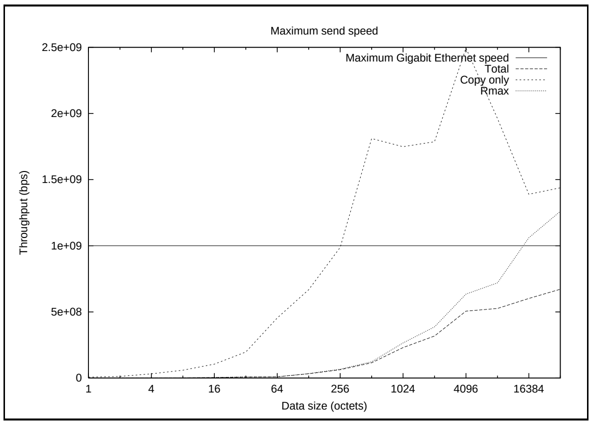

Figure 1. Maximum send speed

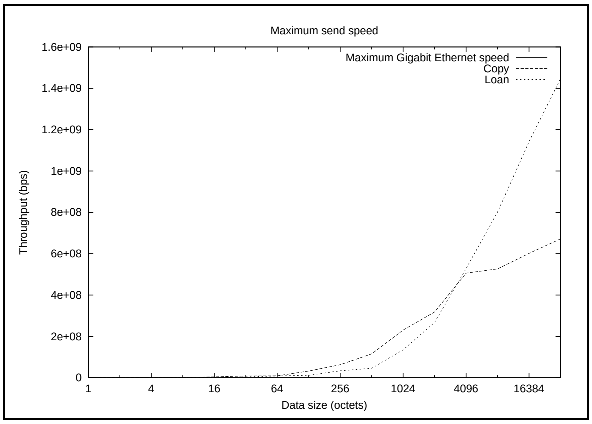

Figure 2. Data copy, page loan comparison

measured overhead from copy operations it is only possible to achieve the maximum Gigabit Ethernet throughput with a data size of at least 256 octets. The highest value is for a 4096 octets data size. The overhead from data copies alone does not appear to prevent communication at gigabit speeds.

The *Total* and R_{max} graphs show the potential improvements from a complete elimination of the overhead from copy operations. It can be seen that data movement is not the only operation performed during protocol processing, but eliminating the overhead from the data copy would result in a substantial performance improvement for larger data sizes. The overhead from the data copy operation starts to matter for data sizes above 512 octets. The R_{max} value increases steadily as the data size increases.

To achieve a high throughput it is clearly necessary to use a large data size, but for larger data sizes the overhead from data copy operations becomes significant.

A comparison of the estimated throughput achievable with *send()*, using either UVM loan or normal data copies

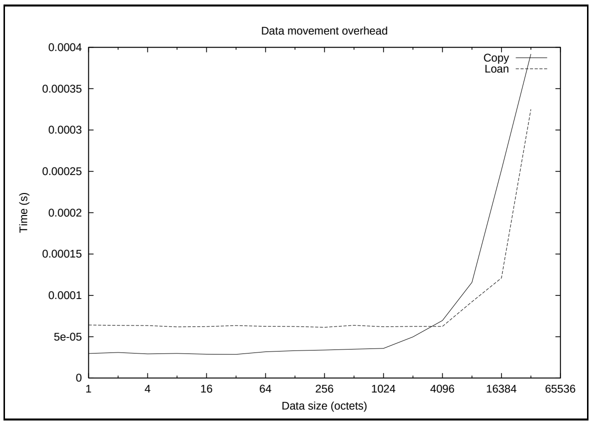

Figure 3. Data copy, page loan overhead

is shown in Figure 2. The page loan mechanism is faster for data sizes of page size (4096 octets) and larger, but copy operations are faster for data sizes below this value. When the data spans multiple pages there is a substantial difference.

A comparison of the overhead of the loan and copy operations alone is shown in Figure 3. The cost of VM operations increases with the number of pages; the overhead from page loan is constant up to 4096 octets. Size starts to matter for copy operations at 1024 octets, and for page loans at sizes greater than 4096 octets. There is clearly a per-page cost for page loan, but the copy cost is larger. The overhead from copy operations is larger than the overhead of the UVM loan mechanism if the data elements are larger than approximately 4096 octets.

3.3. UDP throughput

The section above looked at the overhead from different data movement mechanisms. This section describes the measurement of the maximum achievable throughput during sustained data transfers.

Based on the estimates in Figure 2 it should be possible to saturate the Gigabit Ethernet between the test machines when the page loan mechanism is used.The measurements only estimate the maximum limit for transfer speed based on the overhead from sending a single packet (measured multiple times). How well these calculations compare to sustained transfer is examined with *netperf*. A test run indicates that send errors are frequent. The error count is included in the final results in addition to the throughput.

The result from *netperf* is shown in Figure 4. The performance of UVM loan is better, with an increase of up to 45.64% with a 8 KB data size, but not as good as expected. Instead of increasing with the data size, the throughput decreases for data sizes larger than 8 KB. The estimated maximum throughput and the actual results are shown in

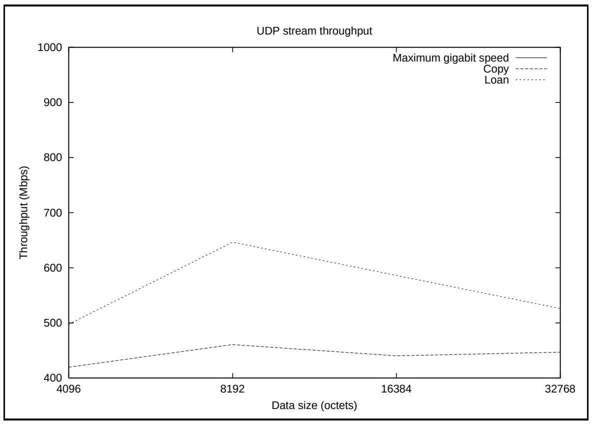

Figure 4. Sustained UDP throughput comparison

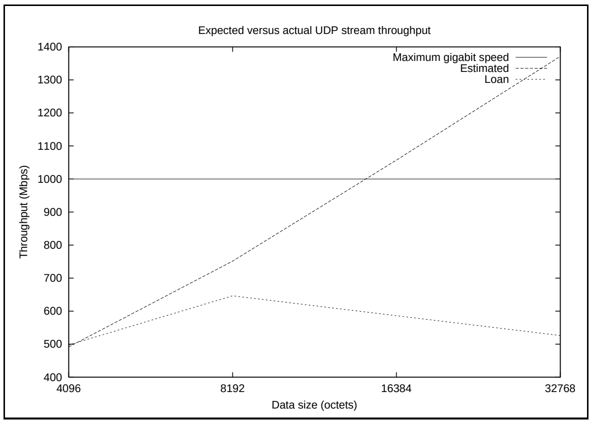

Figure 5. Actual and expected results

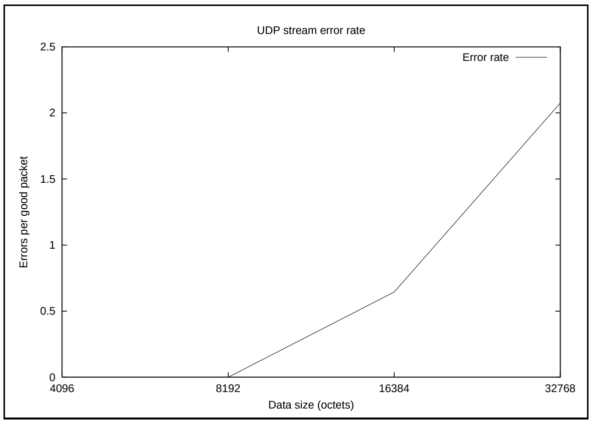

Figure 6. UDP send errors

Figure 5. For 4 KB the measured value is almost as expected, for 8 KB it is slightly lower, but it falls for values above this.

This appears to be caused by a large number of transmission errors, as seen in Figure 6. With a data size of 32 KB there is a significant amount of errors; more than two per successfully transmitted packet. This results in lower performance since much time is wasted on packets which are never successfully transmitted.

The problem turns out to be that the NIC send queue is frequently full. The driver sets the queue size used to the maximum, which is 512 packets, but during heavy network load it is frequently filled. The use of a large data size (32 KB) and a MTU of 1500 octets results in a lot of fragments which quickly fill the output queue.

The machine uses a 32 bit PCI connection to the NIC, and this might be the bottleneck which reduces the performance. We have not been able to verify this yet however. The *netperf* test also indicates that the CPU usage is 100% for both normal copy and UVM loan. A faster CPU might also help address this problem.

Setting the MTU to 9000 octets increases the speed to 954 Mbit/s and reduces the problem, but does not eliminate it. The effects of changing the MTU has however not been properly examined here. A machine with a 64 bit PCI bus might reduce the problem.

In [6], the measured bandwidth of a *null protocol*, which only discards data after moving it to the kernel, reached nearly 800 Mbit/s with UVM loan where it flattened out with a data size of roughly 100 KB. Using a null protocol means that the effects of protocol processing and packet transmission is not experienced. Our results fall after reaching 8 KB instead. The likely reason, as stated above, is that the bandwidth to the NIC has become the major bottleneck. The results from the null protocol in [6] are more likely to represent the maximum achievable and show that the UVM loan mechanism is an efficient data movement mechanism for large buffer sizes.

The maximum throughput value is reached with a buffer size of roughly 100 KB [6]. Our measurements never exceed 32 KB due to the maximum UDP packet size of 64 KB, including headers, and the maximum throughput value from [6] is never reached.

4. Conclusions and future work

This paper describes several measurements made to examine the impact of copy overhead on network communication. A comparison of an application using data copies and one using the UVM loan mechanism to move data from the application buffer to the kernel found the latter to provide significant performance improvements in some cases.

The test system is not able to take full advantage of the improvements since a MTU of 1500 octets results in heavy fragmentation and a large number of transmission errors as the NIC send queue is quickly filled.

This does not reduce the potential benefits from using the UVM loan mechanism, but identifies another possible performance problem which must be taken into consideration when designing high-speed communication systems.

To benefit from the use of zero-copy movement mechanisms such as VM based page movement, it is necessary to ensure proper buffer alignment in applications. This paper does not focus on these issues, but has shown that the benefits from achieving this can be significant. For use in a media server this might not be a problem, since only a small number of applications might need to be modified.

Some of the results point to the PCI bus as a possible bottleneck. We plan to test this on a newer PC with a 64 bit PCI bus.

References

[1] Alteon Networks, Inc., 50 Great Oaks Blvd. San Jose, CA 95119. *Gigabit Ethernet/PCI Network Interface Card: Host/NIC Software Interface Definition*, June 1999. Revision 12.3.11.

[2] E. Anderson. *Container Shipping: A Uniform Interface for Fast, Efficient, High-Bandwidth I/O*. PhD thesis, Computer Science and Engineering Department, University of California, San Diego, CA, USA, 1995.

[3] D. Banks and M. Prudence. A high-performance network architecture for a pa-risc workstation. *IEEE Journal on Selected Areas in Communication*, 11(2):191–202, February 1993.

[4] D. G. Bobrow, J. D. Burchfiel, D. L. Murphy, and R. S. Tomlinson. TENEX, A paged time-sharing system for the PDP-10. *Communications of the ACM*, 15(3):135–143, Mar. 1972.

[5] D. Cheriton. The v distributed system. *Communications of the ACM*, 31(3):314–333, March 1988.

[6] C. D. Cranor. *M.S.* PhD thesis, Sever Institute Technology, Department of Computer Science, Saint Louis, Missouri, August 1998.

[7] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley. Afterburner. *IEEE Network*, 7(4):36–43, July 1993.

[8] P. Druschel. Operating system support for high-speed communication: techniques to eliminate processing bottlenecks in high-speed networking are presented. *Communications of the ACM*, 39(9):41–51, Sep 1996.

[9] A. Gallatin, J. Chase, and K. Yocum. Trapeze/IP: TCP/IP at near-gigabit speeds. In *Proceedings of the FREENIX Track (FREENIX-99)*, pages 109–120, Berkeley, CA, June 6–11 1999. USENIX Association.

[10] H. Massalin. *Synthesis: An Efficient Implementation of Fundamental Operating System Services*. PhD thesis, Columbia University, 1992.

[11] J. K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *Summer USENIX '90*, pages 247–256, Anaheim, CA, June 1990.

[12] T. Plagemann, V. Goebel, P. Halvorsen, and O. Anshus. Operating system support for multimedia systems. *The Computer Communications Journal, Elsevier*, 23(3):267–289, February 2000.

[13] S.-Y. Tzou and D. P. Anderson. The performance of message-passing using restricted virtual memory remapping. *Software – Practice and Experience*, 21(3):251–267, Mar. 1991.