High-speed ATM networking on low-end computer systems

Werner Almesberger
werner.almesberger@lrc.di.epfl.ch

Laboratoire de Réseaux de Communication (LRC)
EPFL, CH-1015 Lausanne, Switzerland

August 31, 1995

Abstract

The practicability of high-speed communication on low-end systems has frequently been questioned and even less demanding variants of high-speed networking standards have been developed to accommodate for the restrictions of contemporary personal computers. In this paper, architectural aspects of existing PC hardware, ATM network adapters, and operating systems are examined, and in fact, serious limitations are discovered. Primarily memory bandwidth is found to be insufficient to support the number of transfers required by traditional networking implementation designs, plus the number of further accesses required for data processing by the sending or the receiving application. The use of single-copy, a concept well known from higher-end systems, is proposed as a means to overcome the memory bandwidth bottleneck. Not only usage scenarios in which single-copy can be reasonably applied, but also situations in which single-copy would yield only marginal improvements or where performance could even deteriorate are identified. Furthermore, implementation issues, such as locking of shared user pages, are discussed. Finally, the performance of single-copy is tested in an implementation of ATM support on Linux, done at LRC, by measuring uni- and bidirectional AAL5 throughput with different PDU sizes. The measurement results indicate that high-speed communication is feasible on today's low-end systems for applications which are primarily uni-directional in nature, and which respect alignment and access constraints imposed by optimizations like single-copy.

1 Introduction

Low-end computers are becoming a potential platform for high-speed networking, such as 155 Mbps ATM. The bottlenecks encountered there are of the same nature as the bottlenecks which existed only a few years ago in workstations [1]. Section 2 describes common entry-level ATM adapter designs. Then, characteristics of contemporary PCs (as a typical low-end host architecture) are compared with those of workstations in section 3. Memory bandwidth is identified as the major bottleneck for network throughput and approaches to use the scarce resources more efficiently are shown. This is followed in section 4 by measurements on an implementation of one of the described solutions. The paper concludes with a critical analysis of the possibilities and the limits of high-speed ATM networking on low-end systems.

Industry-standard PCs running Linux, a POSIX-conformant freeware operating system, are used in this document as typical examples of today's low-end computers. As an indicator for network performance, primarily throughput was considered. It has to be noted that some applications require minimal latency, which is frequently an antagonistic goal to high throughput.

Only Unassigned Bit Rate (UBR) traffic, carrying Classical IP over ATM [2], with TCP and UDP as transport protocols, and "raw" AAL5, are considered in this paper. Other traffic types, especially Available Bit Rate (ABR), will create more subtle requirements for systems of the future. Other important ATM protocols (e.g. LAN emulation) are similar to IP over ATM as far as this study is con-

cerned.

2 Adapter designs

This section briefly describes bottlenecks found in ATM adapters themselves and then describes two competing designs which are very common among entry-level ATM adapters.

2.1 Adapter bottlenecks

Early ATM adapters often suffered from an overly simplistic design. One rather common restriction was that AAL processing had to be performed in software. Although some optimizations are possible even with only marginal hardware support, performance typically remains significantly below the bandwidth offered by the network, and an overly large amount of CPU time is consumed just for using the network.

However, the opposite extreme is also possible. [3] illustrates how designs that attempt to offload all processing to the ATM adapter may yield non-optimal interfaces and that the processing capabilities of the adapter itself may become the new bottleneck.

Modern entry-level ATM adapters therefore perform AAL5 and sometimes also partial AAL3/4 processing in hardware while still remaining comparably simple devices.

2.2 Buffering adapters

The traditional approach for receiving found in many 10 Mbps Ethernet adapter designs is to copy the whole packet from the network to a buffer on the adapter, to interrupt the host when reception is complete, and eventually to copy the packet either under host control ("programmed IO", "PIO"), or using bus-master Direct Memory Access (DMA) to host memory. In the send direction, the corresponding reverse operation is performed. Buffer memory typically has a size of about 8 kB on such Ethernet adapters. For 155 Mbps ATM, more is needed, e.g. 512 kB. A schematic drawing of a buffering ATM adapter is shown in figure 1.

Applying this concept to ATM turns out to be problematic. First of all, AAL5 payloads can be more than 40 times larger than 10 Mbps Ethernet

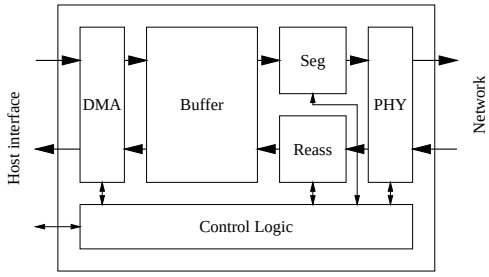

Figure 1: Buffering ATM adapter.

frames (65535 vs. 1500 bytes), so the time needed to transfer a PDU from adapter to host memory may have a noticeable influence on end-to-end delays. As an example, a maximum size AAL5 PDU (64 kB) is sent as one burst at link speed over a 155.52 Mbps ATM link (approximately 16 MB/s effective data rate) in 3.9 ms. If it has to be copied to and from buffers over a 50 MB/s bus at both sides, the total transfer time is increased by 64% to 6.4 ms.

The second problem is that a large amount of expensive buffer space may be required to hold incomplete PDUs of multiple incoming connections. If ring buffers are allocated in contiguous memory (e.g. [4]), each receive VC must have some individual buffer space. Also, send VCs may need separate buffers. Combined with typical hardware restrictions, e.g. that the size of a ring buffer must be an integer power of two, up to 256 kB of buffer space may be needed for each bidirectional ATM connection if maximum size AAL5 PDUs have to be handled.[1]

A comparably large number of interrupts is generated: When sending, the host is interrupted when the packet has been copied to buffer memory, and it is interrupted a second time when the last cell of the packet has been sent.[2] When receiving, the first interrupt is generated when the PDU has arrived in buffer memory. The host then has to determine where in host memory the data should be copied, which may involve some non-negligible processing in the interrupt handler. Finally, the host is inter-

[1] Fortunately, this case is rare. A very common payload size is 9188, corresponding to the default MTU for IP over ATM ([5]).

[2] This interrupt can be used to wake up processes waiting for adapter buffer space.

rupted a second time when all of the data has been copied to host memory.

Nevertheless, the buffering approach keeps the general adapter design simple and may also be used to implement more efficient receive procedures, which may require sophisticated methods to predict traffic properties on adapters doing just-in-time DMA, as described in section 3.5.

2.3 Just-in-time DMA adapters

An alternative approach is to transfer data immediately to host memory when it is received ("just in time"). This eliminates the need to provide large buffers on the adapter and it also reduces end-to-end delays. A schematic drawing of an adapter doing just-in-time DMA is shown in figure 2.

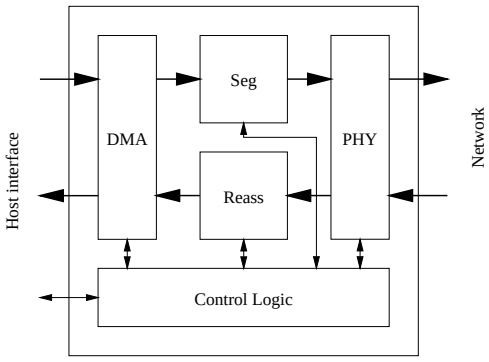

Figure 2: ATM adapter using just-in-time DMA.

In the transmit direction, packets are enqueued in one or more queues in host memory (there is typically one queue per traffic shaper), the adapter copies data using bus-master DMA as cells are being prepared for sending, and an interrupt is generated after the last cell of a PDU has been sent. This process differs from buffer-based approaches basically only in the avoidance of one interrupt and the absence of the copying delay.

In the receive direction, things tend to be more complex. Typically, the host sets up one or more lists of free packet buffers in host memory and replenishes them later on in an asynchronous way. The adapter fetches new buffers from this list when cells are received. After receiving the last cell of a PDU, the host is interrupted. Again, only one interrupt per PDU is necessary. However, additional interrupts may be generated whenever the size of a free buffer list falls below a certain threshold.

This design approach avoids adapter buffer space bottlenecks, but adds the following two problems: first, since only a limited number of cells can be queued on the adapter in the receive and transmit direction, the system architecture must limit the delay until bus master access is granted. Second, alignment and size of receive buffers are determined before the data is received. This complicates some optimizations, as described in section 3.5.

3 Host bottlenecks

Traditionally, the first categories of end systems for which ATM was used in research and industry were typical server- and workstation-type machines. Normally, such systems differ from low-end systems (e.g. industry-standard PCs) primarily in the following aspects:

- better CPU performance, especially for floating-point operations
- frequently: multiprocessing-capability
- higher IO bus bandwidth
- higher memory bus bandwidth

Since ATM is still a relatively new technology, ATM device drivers are typically not optimized to exploit multiprocessing capabilities. Also, modern PC-type machines offer integer processing performance comparable to typical workstations. Differences in CPU power therefore tend to have only a minor impact on network performance.

IO bus bandwidth used to be a major limitation of low-end systems, with theoretical peak bandwidths as low as 8 MB/s (ISA) or 30 MB/s (EISA). Fortunately, the PCI bus, offering workstation-class throughput (133 MB/s theoretical, a little more than 50 MB/s measured) is now being used in all but the cheapest PCs [6]. Since 155 Mbps ATM only transfers about 16.8 MB/s of user data unidirectionally, IO bus bandwidth is rarely a limiting factor anymore.

Table 1 shows characteristic bandwidths of busses in modern PCI systems [7]. The combined bandwidth is the estimated bandwidth for send

Configuration	Neptune Cache	Triton EDO+Cache
Mem to mem	19	39
Mem to IO	51	53
Combined	13.8	22.5

Table 1: Bandwidth (in MB/s) of data paths using chipsets found in modern PC architectures.

or receive operations from or to memory if implemented in the traditional way (see below), i.e. for transfers which involve a memory to memory copy, followed by a memory to IO copy, or vice versa. The effectively available bandwidth can be higher if both types of transfers can – at least partly – be performed simultaneously.

Most operating systems copy network data at least twice: between the network adapter and a kernel buffer, and between the kernel buffer and user space. As shown above, the bandwidth available for this type of operation on common PCs (e.g. using the Neptune PCI chipset) is far below the unidirectional peak data rate of 155 Mbps ATM. Even on the type of PCs which constitute the state of the art at the time of writing, the bandwidth available in the system could not support two simultaneous streams (e.g. copying from a disk to the network) if data is copied to an intermediate buffer.

To summarize, the most critical bottleneck in today's low-end systems is memory bandwidth [8]. Throughput can be improved by reducing the number of times data is copied in host memory. Concepts to achieve this reduction are discussed in the following sections.[3]

3.1 Checksum avoidance

Before any single-copy can be considered, it has to be determined whether an operation with an overhead similar to copying needs to be performed anyway. In TCP/IP protocols, this is frequently the case, since UDP optionally uses a checksum and TCP even requires it. Since on many architectures, checksum computation does not add much overhead if interleaved with a copy operation ([11]), single-copy is not practical in such cases.[4] Note that current entry-level ATM adapters do not support generation or checking of IP, UDP or TCP checksums in hardware.

While interoperability with standard TCP implementations would have to be sacrificed when removing checksums from TCP, no compatibility problems arise for UDP. Since most APIs hide information about the checksum of incoming UDP datagrams from applications, the kernel can transparently omit UDP checksums without the need for application support.

UDP checksums guarantee end-to-end consistency. Because AAL5 provides a much stronger protection – particularly against bursts of errors – by a 32 bit CRC, end-to-end consistency is preserved even without UDP checksums if one ATM connection is used end-to-end. Therefore, UDP checksums are not needed if source and destination are communicating directly over ATM, which typically can be assumed if both machines connect to the same logical IP subnet (LIS, see also [2]) and if that LIS or the destination machine is also the next hop in the sender's routing table.

Figure 3 illustrates the cases where the sender and the destination are on the same LIS, where both are on the same ATM network but in different LISs, and where they are on different ATM networks only connected by an IP router.

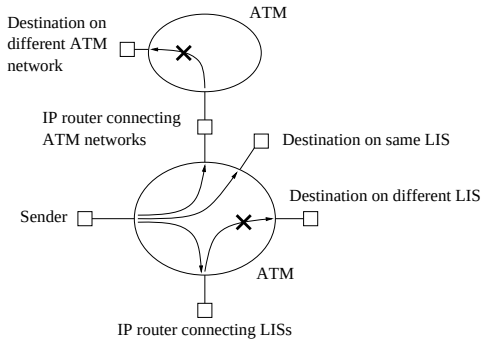

Figure 3: End-to-end consistency is not guaranteed if UDP checksum is omitted on paths crossed out in the picture.

Attempting to remove IP header checksums is

[3] See also [9] and [10] for several approaches to reduce the impact of copy operations on system performance.

[4] However, since computing only the checksum is less expensive than copying data, some minor improvements are possible by avoiding half of the copy operation.

not useful, because that would also cause interoperability problems, and because IP headers are typically too small (20 bytes) to justify any elaborate single-copy concept.

3.2 Send locking

Single-copy in the transmit direction has to lock and protect the pages of the sending application in physical memory, to translate their virtual addresses to physical addresses (and to record them in a data structure called a scatter-gather vector), to set up the transfer, and to unlock the pages when the packet has either been copied to the adapter's buffer or when its last cell has been sent, respectively (figure 4).

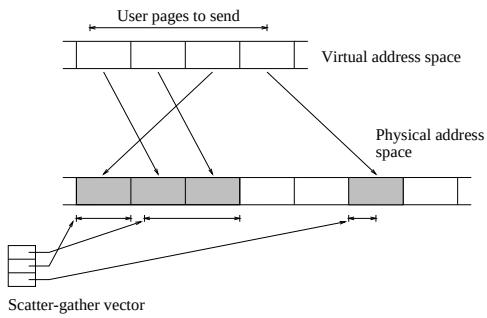

Figure 4: Pages drawn in grey are locked in physical memory.

Performance degradation may occur if the application modifies pages immediately after the send operation, thereby causing the locked pages to be replaced by (writable) copies in the application's address space. If the modification is the result of another system call (e.g. a receive operation), that system call can be extended to unmap pages that will be entirely overwritten by it before writing data.[5] This operation conforms to the semantics of the `read` system call as specified in [12]. It is illustrated in figure 5.

Note that only pages that will be replaced in their entirety can be unmapped. Therefore, performance is improved if buffers are aligned to page boundaries and if their size is a multiple of the physical page size.

[5] If such a page is shared by multiple processes, it has to be replaced in all of them.

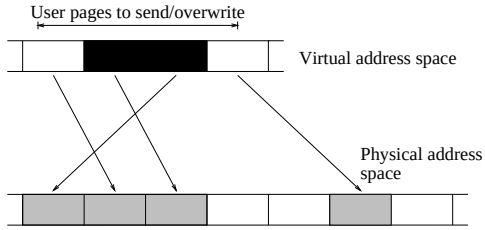

Figure 5: A read operation following a send operation can unmap pages drawn in black from user address space to avoid unnecessary copy-on-write.

A different approach of avoiding excessive copy-on-write overhead would be to block the sending process until the data has been sent or copied. This avoids complicated memory management operations[6] and can also be used in cases where pages are overwritten shortly after sending by something other than a system call. The drawback is that processes may be blocked for a long time if the adapter does not provide a buffer or if that buffer is full.

3.3 Receive prediction

The problems on the receiving side are more varied than for sending. A major difference is that the receive operation may not have been initiated when data arrives, so the ultimate destination in host memory is not known and data has to be buffered, which is typically done in kernel memory.

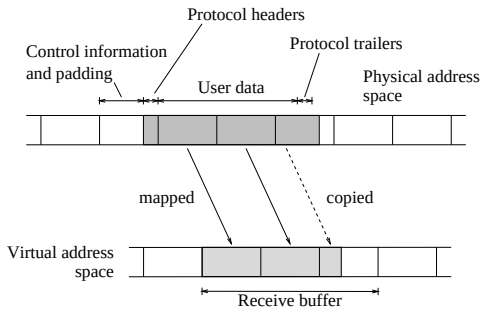

Figure 6: Mapping of memory pages from receive buffers to user space.

In order to avoid the copy operation from ker-

[6] If the memory area is shared with other processes, the effect of concurrent write accesses is undefined.

nel memory to user space, pages of receive buffers can be mapped into user space as illustrated in figure 6. For this to work, the alignment of the data in kernel buffer and user memory has to be the same. Since the semantics of the `read` system call ([12]) and related system calls do not allow the operating system to provide or alter the start address of receive buffers in user space, heuristics have to be used to predict the parameters influencing both alignments:

- size of control information
- size of protocol headers
- placement of the application's receive buffer

Control information is usually constant in size. Since memory allocation inside the kernel is frequently constrained (e.g. to page boundaries), padding bytes are inserted between the control information block and the beginning of data received from the network in order to improve alignment.

The size of protocol headers and the placement of the application's receive buffer usually depend on which application will receive the packet. When several protocols are multiplexed over a single ATM connection (e.g. TCP/IP), heuristics must be used to select the application and the protocol. If the choice was incorrect, the packet still has to be copied from kernel to user space.

Applications may attempt to cooperate[7] by aligning their receive buffers to page boundaries.[8] In this case, the placement of the application's receive buffer is known and only the size of the protocol headers and the receiving application have to be determined. Note that the same information may also be available independent of alignment of receive buffers if the application issues the receive system call before the kernel receive buffer has to be set up.

[7] [13] proposes a mechanism to communicate alignment requirements to applications.

[8] The example suggests that a different alignment may conserve memory by avoiding the need for padding between control information and the received data. While this is true, different alignment constraints may apply to other channels on which the data may travel before or afterwards (e.g. from/to a disk), which typically limits the number of compatible choices.

3.4 Perfect receive prediction

Perfect prediction of the protocol and of the receiving application can be comparably easy with buffering adapters. Since the packet is already present in adapter memory, a quick header analysis can provide all the information needed. Adapters like the Efficient Networks ENI155P used in this case study provide the necessary functionality.

In order to determine the destination of a UDP[9] packet at an end system, only the following fields of an IP packet have to be examined:[10]

- the fragment offset[11]
- the IP protocol number
- the UDP destination port number

After receiving the packet, additional fields (like the IP header size, to determine the presence of options) have to be checked. If their values do not correspond to what was assumed, the usual, less efficient processing has to be performed on the packet.

In addition to the information obtained for application selection, either the UDP packet length or the AAL5 payload length is needed to allocate the appropriate amount of buffer memory.

3.5 Statistical alignment

Single-copy is much more difficult for adapters transferring received cells directly to user memory. They typically fetch free buffers from one or more lists which are replenished by the host asynchronously with respect to the receive operation. The list to get free buffers from is either selected by examining the growth of the incoming PDU (Fore, see [14])[12] or by a static association with VCs (μPD98401, see [15]).

Since there is no possibility to examine packet headers before data is received in host memory,

[9] As explained in section 3.1, TCP is less suitable for single-copy approaches.

[10] Since this is done in an interrupt handler, the amount of processing has to be minimized.

[11] Since fragmentation may be very rare for high-bandwidth traffic on ATM networks, this check can also be deferred until the packet is copied to host memory.

[12] The beginning of a packet is normally stored in a "small" buffer. If the packet is bigger than a small buffer, it is continued in one or more "large" buffers.

alignment choices have to be based entirely on heuristics. Also, the use of free buffer lists delays any reaction to changes in traffic patterns.

Because the μPD98401 only supports buffers of a constant size per free list and because the free list is chosen based on the VC, buffers of worst-case size have to be allocated if data is to be stored in contiguous memory. Freeing the unused part of such a large buffer after the size of the received packet is known may lead to internal memory fragmentation.

Better allocation granularity can be achieved by using another feature of that chip, which is similar to the behaviour of Fore adapters: if one buffer fills up, the next buffer is automatically fetched from the free buffer list.[13] This can be exploited for enqueuing single pages as free buffers, possibly with different alignments in different free lists, see figure 7. In order to avoid fragmentation because of alignment mismatch in adjacent buffers, multiple buffers with the same alignment should be allocated back to back.

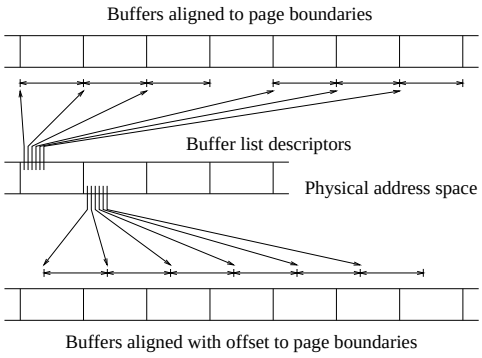

Figure 7: Alignment of buffers in free lists.

A problem arises when protocol headers or trailers do not appear on adjacent pages, because protocol processing code should not be burdened with the requirement to perform complex memory address translation for each access. For headers, the problem can usually be avoided by making buffers large enough for the largest expected header (whose size can normally be estimated) to fit. Since headers rarely grow beyond typical memory page sizes, no special precautions may be necessary.

Unfortunately, trailers may be placed at about any address, so they need to be treated separately. As for headers, the maximum size of a trailer can be estimated. After receiving a packet, the kernel can check whether the whole trailer is on the last page.[14] If this is not the case, two new, contiguous pages are allocated and the content of the two last pages is copied to these new pages. Since this operation should be very infrequent, no major performance loss should occur.[15]

This concept can also be extended to handle cases where the maximum header or trailer sizes are not known in advance and where new information about encapsulation is obtained incrementally, e.g. at different protocol layers.

4 Experimental results

Support for sending and receiving "raw" AAL5 datagrams over the socket interface with or without single-copy has been implemented in the Linux kernel at LRC.[16] IP over ATM performance will be studied in the future.

Performance measurements were done on PCs with 90 MHz Pentium processors, 32 MB of RAM, a PCI bus using the Intel Neptune chipset, and Efficient Networks ENI155P-MF-C 155 Mbps OC3 ATM adapters. Send and receive buffers in the adapters were set to 128 kB, buffers in the kernel were not bounded. A version of the `ttcp` program with extensions to support ATM was used for the measurements.

In addition to raw AAL5, partial support for Classical IP over ATM ([16]) using LLC/SNAP encapsulation ([17]) over PVCs has been implemented.

[13] The actual organization is even more complex since buffers are grouped in batches. Batches correspond to packets, and the overflow mechanism is applied in this form to batches only. However, since a batch may contain only a single buffer, it can be said that overflows with a granularity of a single buffer are possible.

[14] This assumes that the maximum trailer is not bigger than one page. If trailers can grow beyond the size of a single page, the numbers have to be increased accordingly.

[15] If trailer fragmentation occurs systematically on a VC, a different memory allocation strategy has to be chosen for that VC.

[16] More information about the current implementation can be obtained from `http://lrcwww.epfl.ch/linux-atm/`

4.1 Unidirectional transmission

Two PCs were used for this experiment. One of the PCs sent a burst of 2048 datagrams of constant size from user memory to the other PC, which received them to user memory. No additional data processing was done. Both PCs were connected directly, without a switch.

Without single-copy

Figure 8 shows sender and receiver data rates at the socket interface. The theoretical maximum data rates for AAL5 traffic on 155 Mbps OC3 ATM (90.4 Mbps to 135.4 Mbps) are also indicated.[17] Note that the send data rate can be higher than the effective transmission rate if data is copied faster from user space to kernel space than it is sent over the network. The error bars indicate the extremal values measured in ten experiments.

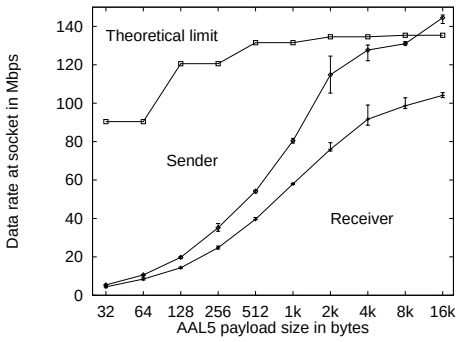

Figure 8: Data rate of "raw" AAL5 traffic at the socket interface when copying to and from receive and send buffers.

As can be seen, the sender is able to come reasonably close to the maximum data rate only for datagrams of 8 kB or larger. The receiver can barely handle 100 Mbps and also requires large datagrams for reasonable performance.[18]

For loss-free continuous transmission, the sender would have to adapt to the maximum rate the receiver can handle.

With single-copy

Much better throughput has been achieved when avoiding copying data from and to the intermediate kernel buffer. The measured data rates are shown in figure 9.

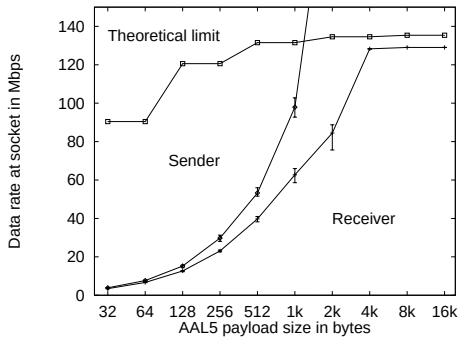

Figure 9: Data rate of "raw" AAL5 traffic at the socket interface when using single-copy.

There is no difference to the copying approach in the receive direction for datagram sizes below the memory page size, 4 kB, because the contents of partially filled pages are always copied. As soon as single-copy can be used, the data rate immediate jumps to approximately 128 Mbps and increases for larger datagrams only to 129 Mbps.

In the send direction, pages are always locked in physical memory, independent of the amount of data transferred. This operation is slightly more complex than allocation of a buffer and copying of data, so up to about 512 bytes, the solution with copying is slightly faster. For 1 kB and larger datagrams, the non-copying version is much faster. Maximum network speed is reached already for 2 kB datagrams. In fact, the possible throughput increases almost exponentially, as is shown in figure 10.

4.2 Loopback transmission

Although the unidirectional tests illustrate how single-copy improves the throughput, they do not correspond to typical real-life applications, where

[17] For clarity, only the rates for the datagram sizes used in the measurements are shown. The rates for all possible sizes have larger variations, because of the fixed cell size.

[18] One of the reasons why the receiver is significantly slower is that the device driver uses different DMA burst sizes for receiving and sending. This had to be done to work around a problem with the Neptune chipset.

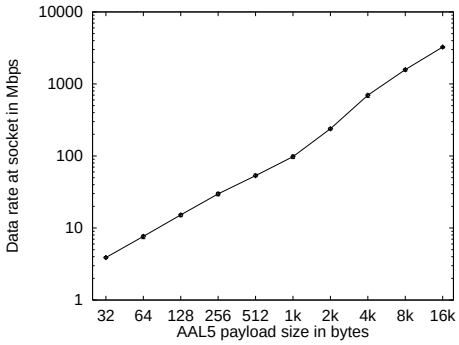

Figure 10: Send (enqueue) rate at the interface when using single-copy.

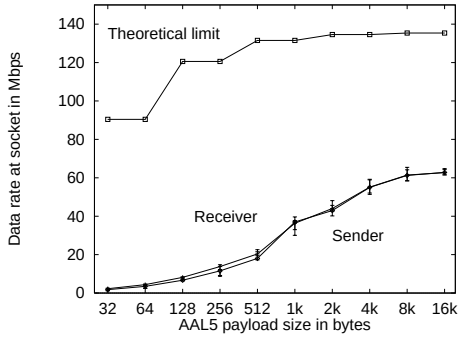

Figure 11: Data rate of "raw" AAL5 traffic at the socket interface when copying to and from receive and send buffers.

data is generated or loaded at the sender, and processed, displayed, or stored at the receiver. This means that at least twice the network bandwidth is needed inside the system to perform some meaningful tasks.

A very simple test has been designed to obtain indicative results for this case: one PC runs a sending and a receiving `ttcp` on the same VPI/VCI. An external loopback is formed with the fiber.

Without single-copy

Figure 11 shows the case using buffering. Not unexpectedly, the sender and the receiver run at approximately the same speed and the total throughput hardly exceeds 60 Mbps, which corresponds roughly to half the maximum bandwidth calculated for the entire system.

It should be noted that in most cases, the receiver was faster than the sender. The reasons for this paradoxical behaviour are probably variations in when timing starts, i.e. the receiver starts its timer after receiving a small start datagram, which happens at a time when the system is probably already overloaded by the sender and the device driver.[19]

With single-copy

Again, much better performance was measured when using single-copy, as shown in figure 12. For datagram sizes below 1 kB, the throughput is worse than in the copying case, because of the increased processing overhead. For 1 kB and 2 kB, throughput is already increased, because single-copy in the sender loads the system less. Finally, at 4 kB and more, the receiver can benefit from single-copy too, the memory bottleneck disappears, and data flows at full speed. The system appears to be saturated at a maximum receive speed of 115 Mbps.

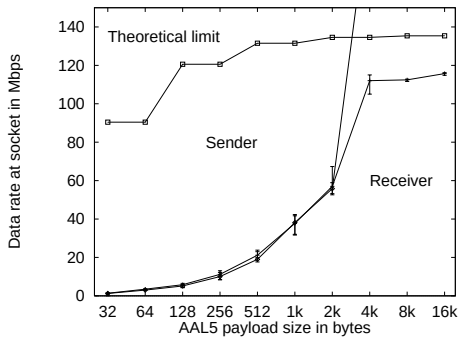

Figure 12: Data rate of "raw" AAL5 traffic at the socket interface when using single-copy.

5 Conclusion

Memory bandwidth has been identified as the main bottleneck for high-speed networking on current low-end systems and the use of single-copy has been determined to be a straightforward yet effective way of overcoming this bottleneck in many

[19] Also, the less efficient DMA burst mode of the receiver now slows down both parties.

cases. More adaptive approaches promise to solve also some of the problems associated with single-copy (e.g. complexity and limited use).

We have implemented basic ATM support on Linux and obtained experimental results for AAL5 traffic. Even in this simple implementation, avoiding to pass network traffic through an intermediate buffer has significantly improved the throughput that can be achieved on low-end systems.

Bidirectional throughput in current low-end systems is still not sufficient to saturate the network, and actual processing of bidirectional high-speed streams would even require more throughput. Further work is required to overcome these limitations.

References

[1] Druschel, P.; Abbott, M.; Pagals, M.; Peterson, L. *Network subsystems design*, `ftp://cs.arizona.edu/xkernel/Papers/analysis.ps`, IEEE Network, vol. 7, pp. 8–17, July 1993.

[2] Cole, R. G.; Shur, D.. *IP over ATM: A Framework Document* (work in progress), Internet Draft `draft-ietf-atm-framework-doc-04.ps`, July 1995.

[3] Basu, A.; Buch, V.; Vogels, W.; von Eicken, T. *U-Net: A User-Level Network Interface for Parallel and Distributed Computing* (CS-TR to appear), `http://www.cs.cornell.edu/Info/Projects/ATM/unet-tr.ps`, Cornell University, April 1995.

[4] Efficient Networks, Inc. *Midway (SBUS) ASIC Specification*, October 1994.

[5] Atkinson, R. *Default IP MTU for use over ATM AAL5*, RFC1626, May 1994.

[6] Schnurer, G. *Moderne PC-Bussysteme*, c't 10/93, pp. 110–119, October 1993.

[7] Schnurer, G. *Wider den Flaschenhals*, c't 4/95, pp. 128–136, April 1995.

[8] Clark, D. D.; Jacobson, V.; Romkey, J.; Salwen, H. *An Analysis of TCP Processing Overhead*, IEEE Communications Magazine, vol. 27, pp. 23–29, June 1989.

[9] Partridge, C. *Gigabit Networking*, Addison-Wesley, October 1993.

[10] Steenkiste, P. A. *A Systematic Approach to Host Interface Design for High-Speed Networks*, IEEE Computer, vol. 27, pp. 47–57, March 1994.

[11] Braden, R.; Borman, D.; Partridge, C.; *Computing the Internet Checksum*, RFC1071, September 1988.

[12] IEEE Standard for Information Technology. *Portable Operating System Interface (POSIX). Part 1: System Application Program Interface (API)*, IEEE, July 1994.

[13] Almesberger, W. *Linux ATM API*, `ftp://lrcftp.epfl.ch/pub/linux/atm/api/`, June 1995.

[14] Fore Systems, Inc. *Programmer's Reference Manual for AALI Interface*, `ftp://ftp.fore.com/pub/docs/aali.ps`, November 1994.

[15] NEC Corporation. *μPD98401 local ATM SAR chip user's manual*, IEU-1384, June 1994.

[16] Laubach, M. *Classical IP and ARP over ATM*, RFC1577, January 1994.

[17] Heinanen, J. *Multiprotocol Encapsulation over ATM Adaptation Layer 5*, RFC1483, July 1993.