Demultiplexing on the ATM Adapter: Experiments with Internet Protocols in User Space *

Ernst W. Biersack, Erich Rütsche
B.P. 193
06904 Sophia Antipolis, Cedex
FRANCE
e-mail: erbi@eurecom.fr, rue@zh.xmit.ch

Abstract

We took a public domain implementation of the TCP/IP protocol stack and ported into user space. The user space implementation was then optimized by a one-to-one mapping of transport connections onto ATM connections and a packet filter. We describe the user space implementation and compare its latency and throughput performance with the existing kernel implementation.

* Published in: Journal on High Speed Networks, Vol. 5, No. 2, May 1996

1 Introduction

Processing of protocols has been perceived as a major bottleneck in the communication performance. To get around this bottleneck various approaches have been examined. The offloading of entire protocols or protocol functions onto adapters or communication subsystems was not successful because the I/O interface of the workstations limited the performance gained by the subsystem. Experiments with newer, so called light-weight protocols showed that improved protocols could add functionality but their performance did not give sufficient evidence to replace traditional protocols such as TCP/IP. However, in the implementation of protocols still a considerable gain can be achieved by better interfaces to the network adapters [DALT 93] and by better integration of the protocol stack and its application.

We investigate *user level* protocol implementation that takes advantage of the de-multiplexing functions being implemented on todays high-speed ATM adapters. A second argument for a user level implementation is to provide the application with a dedicated implementation of the protocol stack that allows better control over protocol processing to guarantee the QoS of the networked data [THEK 93, EDWA 94].

2 Concept

ATM (Asynchronous Transmission Mode) and its Adaptation Layer (AAL) offer a technology for the physical layer and the link layer of high-speed LANs and WANs. The AAL offers a connection-oriented frame transfer service on top of ATM. The AAL functions to reassemble a frame out of ATM cells are simple and must be executed at high speed. Therefore, the AAL is often implemented by dedicated processors on network adapters that offer the AAL service to the host system. Higher layer protocols can be implemented by mapping several higher layer connections onto a single AAL connection or by mapping each higher layer connection onto a dedicated AAL connection (*direct mapping*). This mapping can be done based on the number of possible AAL connections or based on performance and QoS criteria of the higher layer connection and application.

In this paper we examine the advantages of a direct mapping of a transport layer connection onto an AAL connection. We did the experiments with an implementation of the Internet protocols in the user space.

In a normal stack protocol, processing is done in a tree-like way. On each level branches are taken to demultiplex incoming PDUs to the correct higher layer protocol. The sum of these demultiplexing operations can take a considerable amount of time because in each layer the header must be parsed and the right connection control information must be found. While these operations are necessary to open a connection in a protocol stack, the normal dataflow case allows an optimized protocol handling. In the normal case, the protocol headers have the same format, no errors happen and consecutive PDUs for the same protocol belong to the same connection [CLAR 90]. If we assume that a protocol stack of a single connection is mapped directly onto a dedicated AAL connection then additional simplifications for protocol processing can be made.

All address information is known a-priori by the direct mapping. Therefore protocol handling can be simplified to the processing of the parameters that change and to exception handling. The stacked protocol headers are compared to a precomputed filter that looks for exception handling and for precomputed window information. If an exception is detected or if the header does not match the expected format, then the slow standard processing path is taken. If the filter matches an optimized processing path is taken that processes only the necessary steps, e.g., acknowledgments and timers. To allow an optimal matching of the filter the protocol parameters must be set to prevent segmentation.

3 Implementation

3.1 Multiplexed Protocol Stack in User Space

The internet protocol stack has been implemented in two steps. In the first step the code of IP, UDP, and TCP of the 4.3BSD has been ported to the user space of a Sun SPARC with SunOS Version 4.1.3. The goal was to build a library such that any application could chose either the system protocol stack or our library without any change. This implementation still multiplexes multiple transport connections onto a single ATM connection (see figure 1). The ATMcl layer in figure 1 performs the necessary adaptation between the connection-less IP and the connection-oriented ATM. Before the first IP datagram can be sent, an ATM connection is established. The ATM connection is released under timer control when no more IP datagrams are transmitted.

We changed the memory management of the protocol stack to use the mbuf structures on memory blocks allocated via a malloc system call. Our socket interface copies the user data to be sent into these memory structures where the subsequent protocol processing takes place. The protocols build packets by chaining the data and the headers in mbufs. Data to be sent are passed via a standard I/O interface to the AAL that is implemented on a SBA200 board from FORE Systems. A `select` call signals the reception of data on the same interface. Our network driver reads these data and hands them to IP, which forwards the transport layer PDU to TCP or UDP. The socket `receive` call gets the data from the transport layer and copies them to the user buffer. We use the Pthread library [MUEL 93] to implement a parallel timer thread that watches all outstanding timers.

3.2 Optimized Protocol Stack in User Space

In the second step we optimized the user space protocol implementation by a *direct mapping* between a transport connection and an AAL connection and a *packet filter*.

Each TCP connection is mapped onto a different ATM connection. The demultiplexing then takes only place at the ATM layer. This architecture is depicted in figure 2.

The packet filter looks for all the fields in the PDU that can be precomputed once the connection is established. As the addresses are known a priori, the filter matches only the fields that can change, e.g, IP segmentation and options, TCP flags and window,

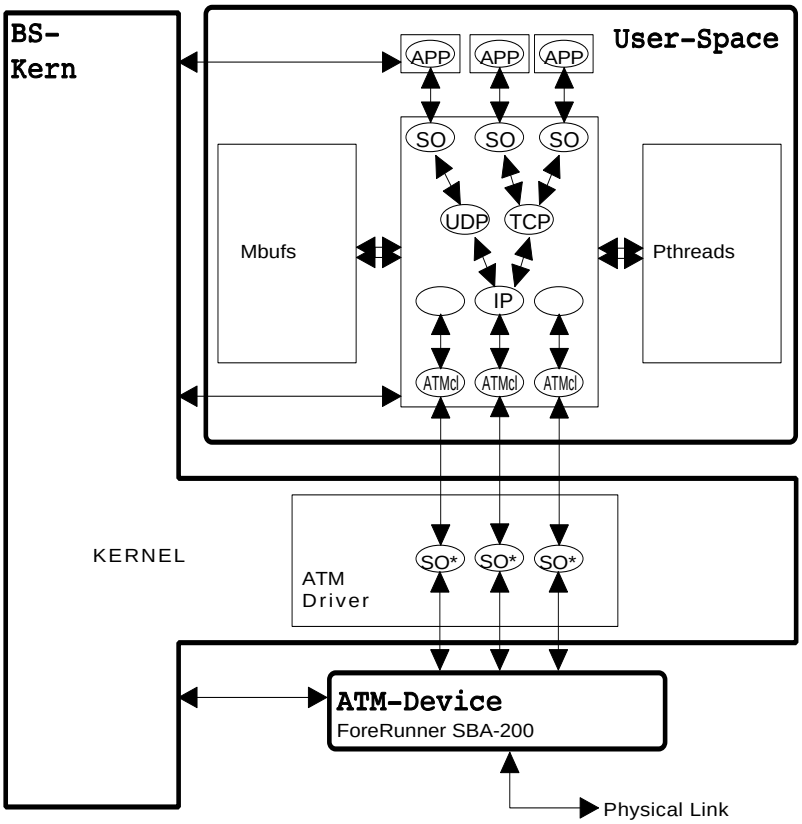

Figure 1: Architecture of the multiplex protocol stack in user space.

UDP flags. The algorithm is close to the header prediction algorithm of TCP by McCanne and Jacobson [MCCA 93] but is enhanced by the filtering of the IP header.

To implement the filter, the code for the receiver side of the IP and TCP / UDP was modified to distinguish between two different cases, the execution of which leads to two different paths (see figure 3):

- If the precomputed filter does not match, the *standard path* is executed that implements the full IP and TCP / UDP and can handle all options.
- If the precomputed filter does match, the *fast path* is executed, implementing a reduced version of IP and TCP / UDP. Here, a number of functions are suppressed, such as connection lookup, option processing, window adaptation, PCB-searching, or IP segmentation and checksum computation.

The decision which of the 2 paths to execute is taken by the filter that checks a certain number of fields in the header of IP and TCP / UDP. For IP and TCP these fields are highlighted in figure 4.

In the IP header, the filter checks if

- VERS == 4, i.e. the current version of IP is used
- HLEN == 20, i.e. the header is 20 bytes long and does not contain any IP options
- SERVICE == 0, i.e. there are no particular QOS requirements

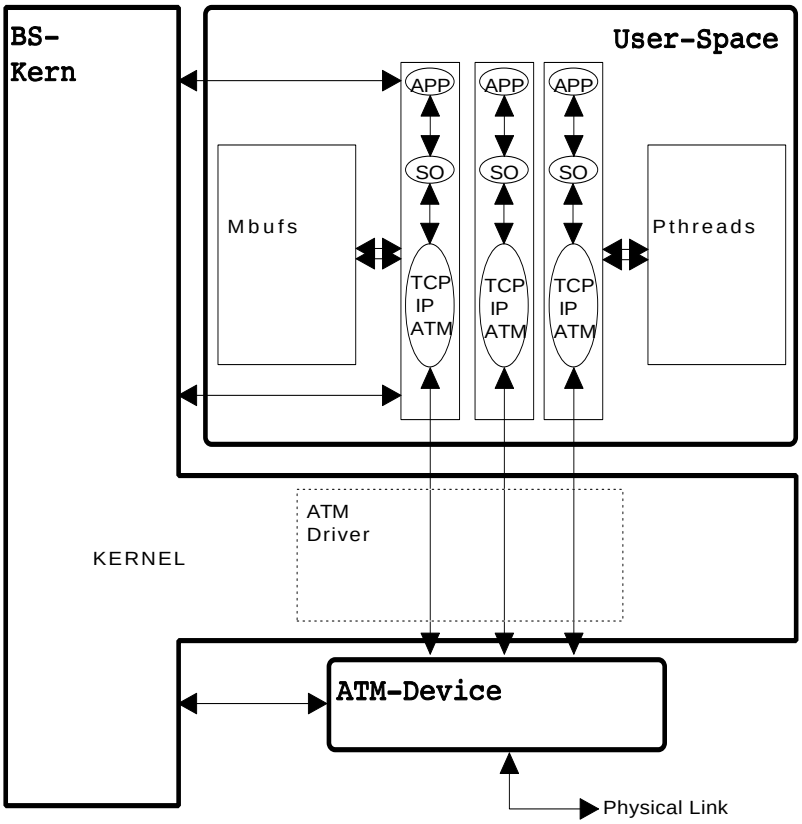

Figure 2: Architecture of the optimized protocol stack in user space.

- FLAGS == 0, i.e. there is no fragmentation
- FRAGMENT-OFFSET == 0, i.e. the IP datagram is not fragmented. We avoid fragmentation by choosing the maximum size for the TCP PDU such that it fits into a single AAL-5 PDU of 4096 Bytes.
- PROTOCOL == 6, to verify that TCP is used as transport protocol.

The filter matches on the TCP header iff

- The header does not contain any options (check of HLEN)
- The Flags URG, SYS, FIN, and RST are not set (check of CODE-BITS)
- The window size was not changed by the receiver (check of WINDOW).

If the standard path is taken, a filter adaptation is necessary when the window size was changed. In this case, the filter is adjusted to retain the new value of WINDOW. When the next PDU arrives, it can again become eligible for the fast path, provided that all the other predicates checked by the filter match.

Figure 5 shows that the fast path allows to suppress a certain number of functions.

When taking the fast path, at IP level

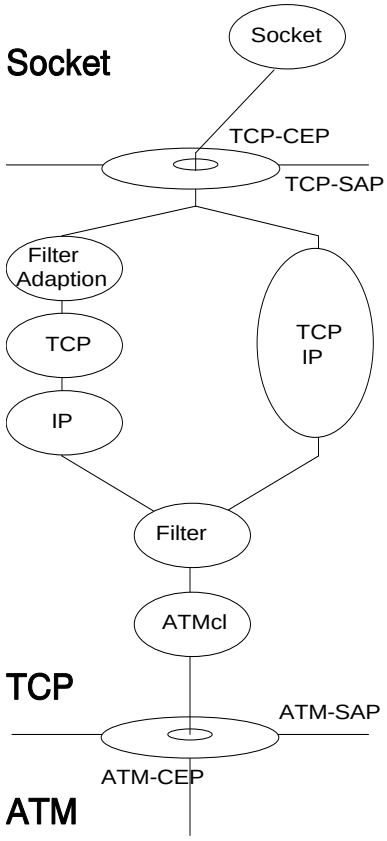

Figure 3: Integrated TCP-IP implementation with filter.

- The header checksum will not be verified since we know that the IP datagram was encapsulated in an AAL-5 PDU that performs itself an error detection – using a 32-bit CRC – that covers the whole IP datagram.
- The address parsing is omitted since there is a one-to-one mapping of the TCP connection onto the ATM connection.

When taking the fast path, at TCP level

- The option processing is omitted
- There is no need to search for the right protocol control block (PCB) because of the one-to-one mapping of the TCP connection onto the ATM connection.

4 Performance Results

For our experiments we used two SPARC-10 stations with SunOS 4.1.3. The workstations are equipped with SBA-200 ATM adapter cards from FORE and connected via an ASX-100 ATM switch from FORE. In the following we will compare the two user level implementations referred to as **MUX**, for the first version that does multiplexing above ATM and **OPT** for the second optimized implementation. The standard kernel level implementation will be referred to as **SYS**.

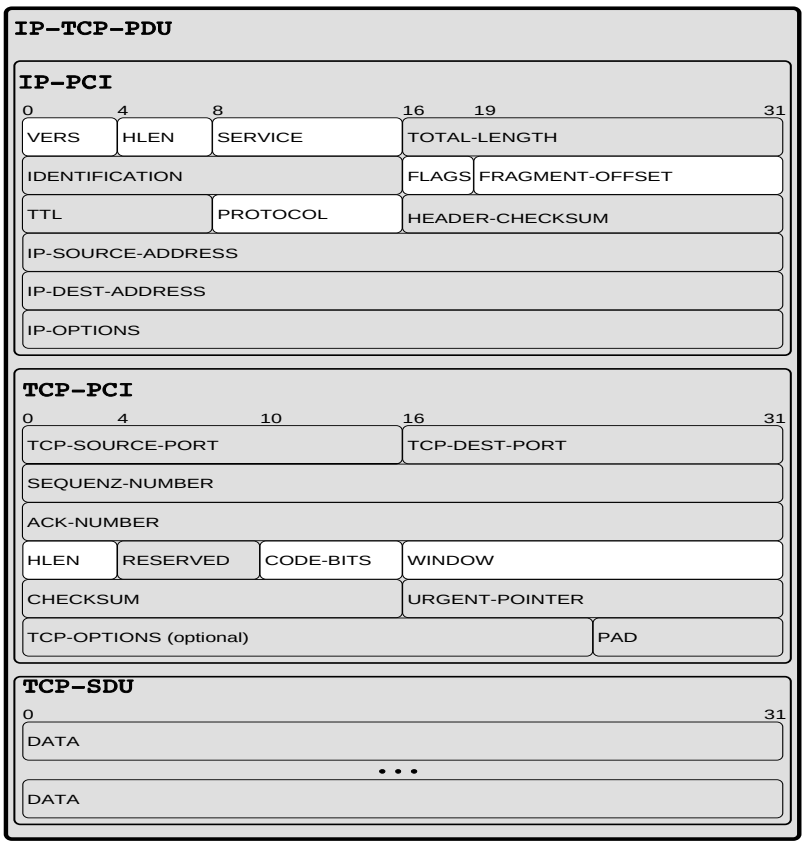

Figure 4: Filter mask for TCP-IP protocol header.

4.1 Latency

In a first experiment we determined the latency improvement for OPT as compared to MUX. We sent a large number of packets with one byte user data from a client to a server and back to the client again to measure the

- Round trip time **RTT**
- Time **C_snd** spent at the client traversing the socket, TCP / UDP, and IP layer
- Time **S_rcv&snd** spent at the server traversing the IP, TCP / UDP, and socket layer up and then down again
- Time **C_rcv** spent at the client traversing the IP, TCP / UDP, and socket layer.

The time **Atm** spent traversing the ATM layer, driver and switch is computed as $Atm = RTT - C\_snd - S\_rcv\&snd - C\_rcv$.

With TCP and UDP as transport protocols we obtained the following latencies (see Table 1 and 2.)

The optimized version OPT achieved a significant reduction of the processing time at the receiver side (C_rcv): 41% with UDP and 20% with TCP as transport protocol.

It is worth noting that only a small fraction of the total RTT is spent executing IP, TCP / UDP, and the socket layer.

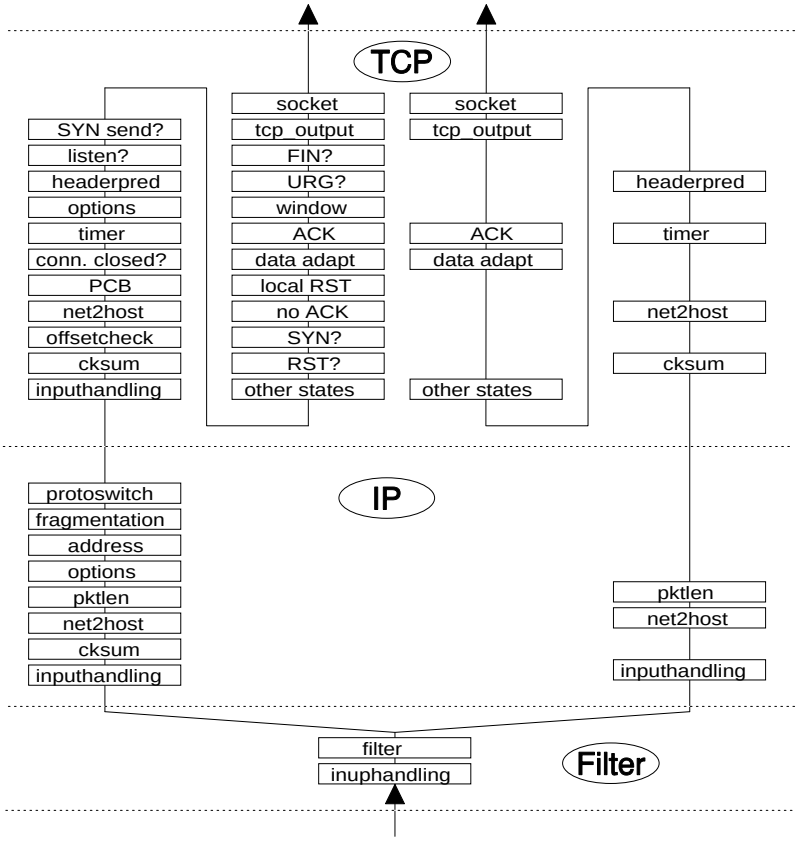

Figure 5: The two different paths in the TCP-IP implementation.

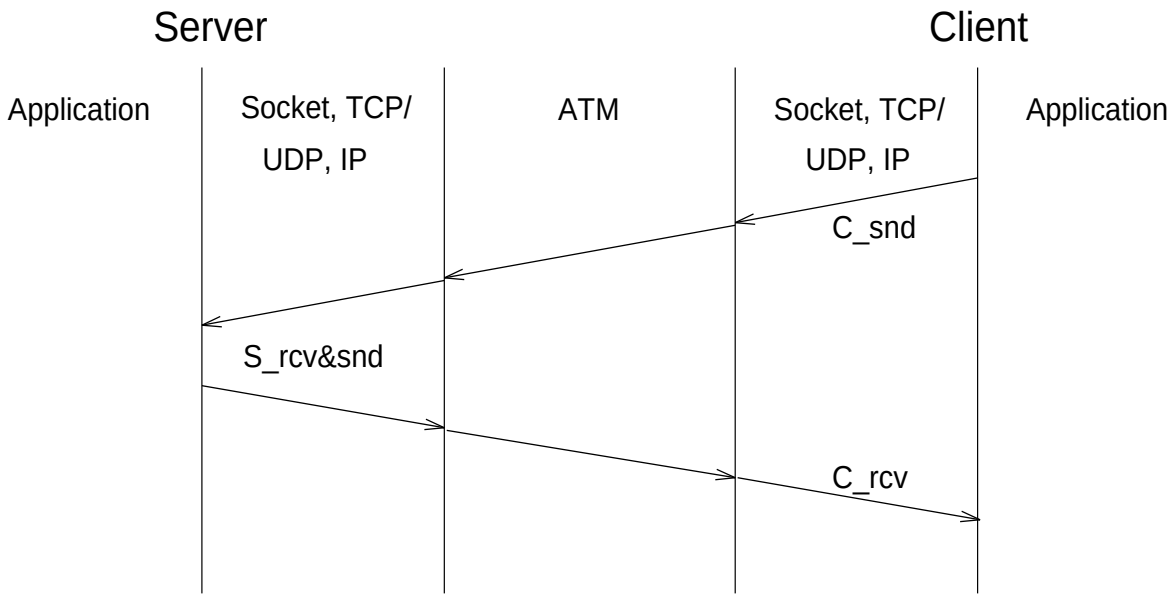

Figure 6: Latency measurement.

4.2 Throughput

The throughput measurements with TCP as transport protocol are given in table 3 and with UDP as transport protocol are given in table 4.

	RTT	Atm	C_snd	S_rcv&snd	C_rcv
SYS	2160				
MUX	2720	1810	177	440	285
OPT	2570	1700	169	387	228
OPT / MUX	95 %	95 %	95 %	87 %	80 %

Table 1: Latency with TCP as transport protocol [μs].

	RTT	Atm	C_snd	S_rcv&snd	C_rcv
SYS	2170				
MUX	2380	1720	145	329	188
OPT	2230	1730	150	238	110
OPT / MUX	94 %	100 %	100 %	72 %	59 %

Table 2: Latency with UDP as transport protocol [μs].

The two user level implementations MUX and OPT achieve about the same throughput, i.e. the optimizations in OPT do not improve the throughput.

The throughput for the kernel implementation SYS is noticeably higher than for the two user level implementations. For UDP with 2 KByte size packets we measure 3.3 MByte / sec for SYS and only 2.5 MByte / sec for MUX or OPT. The lower throughput performance for MUX and OPT is – at least in part – due to the higher number of copy operations, both at the sender and receiver side. With TCP as transport protocol we have for the user level implementation 5 copy operations at the sender side and 3 at the receiver side as compared to 3 and 2 copy operations for the kernel implementation (see figure 7 and 8). The fat arrows indicate physical copies and the thin arrows indicate the passing of buffers via buffer pointers between layers.

If user level implementations want to achieve a throughput comparable to the kernel implementation, one needs to reduce the number of copy operations. This can be achieved by better integrating the user level implementation with the ATM driver and with the applications. Ideally one would move the data from the application to the network adapter with a single copy operation. This is currently not possible since we don't have the sources for the ATM driver.

	Buffer size [Byte]			
	512	1024	1500	2048
SYS	2.16	2.66	1.88	2.70
MUX	1.36	1.71	1.85	1.91
OPT	1.35	1.72	1.83	1.90

Table 3: Throughput with TCP as transport protocol [MByte / s].

	Packet size [Byte]			
	512	1024	1500	2048
SYS	1.22	2.15	2.09	3.30
MUX	0.80	1.48	2.00	2.47
OPT	0.81	1.45	1.99	2.49

Table 4: Throughput with UDP as transport protocol [MByte/s].

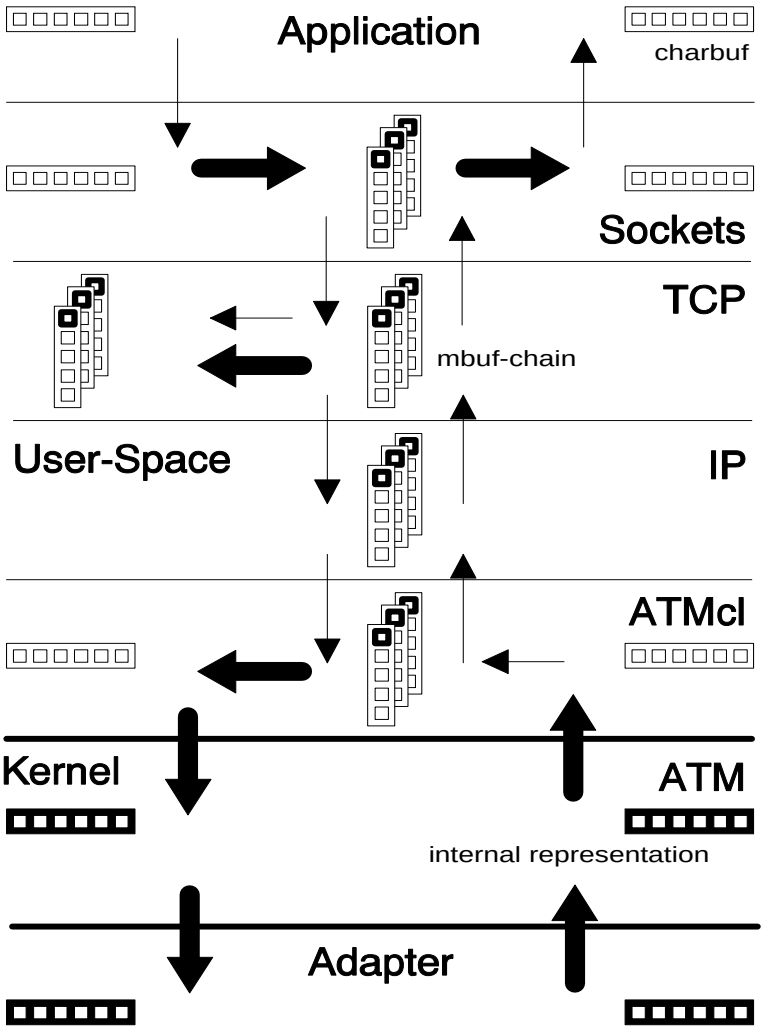

Figure 7: Data copy operations for user space implementation.

Acknowledgement

We would like to thank Thomas Unterschütz for helping us with the experiment.

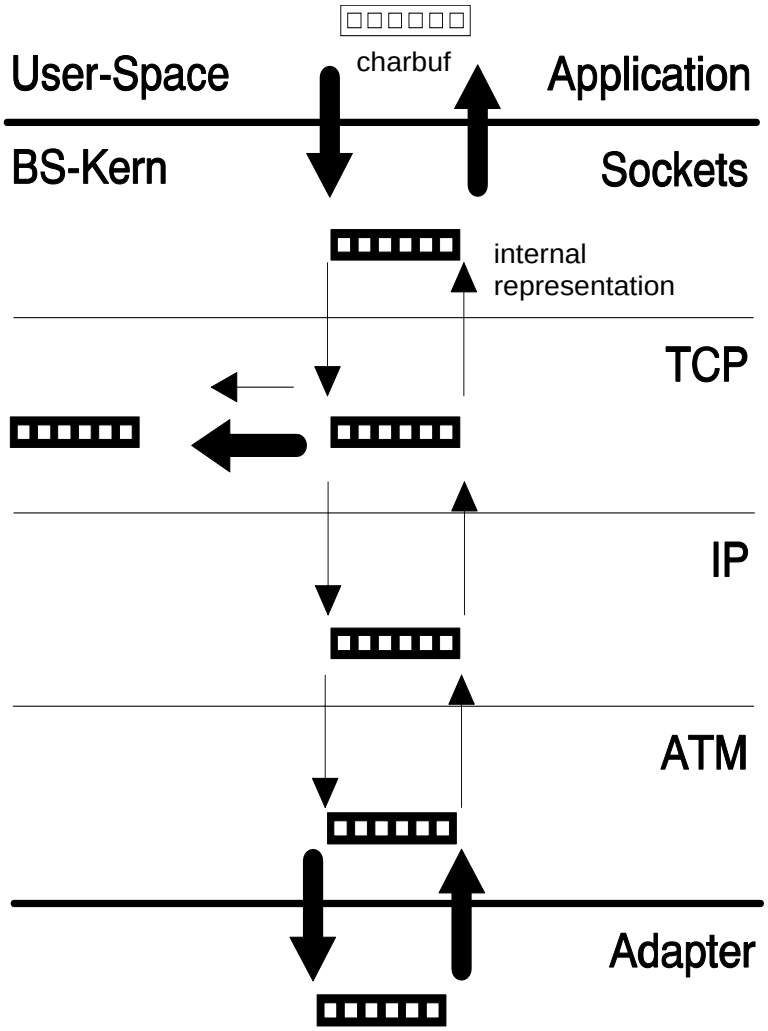

Figure 8: Data copy operations for the kernel implementation.

References

[CLAR 90] D. D. Clark and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proc. ACM SIGCOMM 90*, pp. 200–208, Phildelphia, PA, September 1990.

[DALT 93] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards and J. Lumley, "Afterburner", *IEEE Network*, 7(4):36–43, July 1993.

[EDWA 94] A. Edwards et al., "User-Space Protocols Deliver High Performance to Applications on Low Cost Gb/s LANs", *Proc. SIGCOMM 94*, pp. 14–23, London, England, September 1994.

[MCCA 93] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture", *1993 Winter USENIX*, San Diego, CA, January 1993.

[MUEL 93] F. Mueller, "A Library Implementation of POSIX Threads under UNIX", *1993 Winter USENIX*, San Diego, CA, January 1993.

[THEK 93] C. Thekkath, T. Nguyen, E. Moy and E. Lazowska, "Implementing Network Protocol at User Level", *IEEE/ACM Transaction on Networking*, 1(5):554–565, October 1993.