TCP Performance over ATM on Linux and Windows NT

Martin Borriss Uwe Dannowski
Hermann Härtig
Dresden University of Technology
Computer Science Department
Operating Systems Group

Phone: +49-351-4638401 Fax: +49-351-4638284
Email: {borriss,ud3,haertig}@os.inf.tu-dresden.de

Keywords: Performance, TCP, ATM, Operating Systems

Abstract

In today's local area networks ATM is often used as replacement for slow Ethernet. This work measures and compares performance of the *Transmission Control Protocol* (TCP) over ATM on two popular operating systems on PC hardware. Throughput and round trip latency of TCP data transfer over both Linux and Windows NT on identical hardware were measured. For throughput measurements, send and receive performances have been isolated by using a fast third-party machine as peer.

Firstly, measurements indicated that high-end PC hardware can utilize the bandwidth provided by 155.52 Mbps ATM network adapters well. Running the heavyweight TCP protocol, data rates of up to 83% of the bandwidth available have been observed.

As a second result, Linux and Windows NT bulk data throughput were competitive. However, particularly on slow hardware, the Linux implementation consistently outperformed NT.

Finally, significant latency differences in the order of a 50% advantage for Linux were indicated by the request–response test suite.

1 Motivation

Part of the current work of the Operating Systems Group at Dresden University of Technology deals with operating system support for predictable high-speed networking. The ability to estimate resource utilization by networking protocols and networking applications involves finding upper bounds on possible data throughput and latency, depending on the hardware and protocols used.

Furthermore, since the Linux device driver for the ATM boards used was written in our group, we were naturally interested to verify its competitiveness.

As discussed in Section 2.2, using TCP over ATM was expected to be a most challenging test for operating system, protocol and driver software.

2 Environment

To make the results as expressive as possible, identical machines have been used. All measurement were done on both "slow" and "fast" machines by today's standards. While measuring, no CPU-intense applications ran concurrently on the participating machines.

Additionally, another communication partner, a

Sun-Ultra-1 machine has been included. This was done for two reasons:

- Particular optimizations, such as proprietary TCP flow control, would not go completely undetected in a heterogeneous environment.
- Isolation of send and receive performance is possible.

The test environment is visualized in Figure 1. The next section describes the hardware used in more detail.

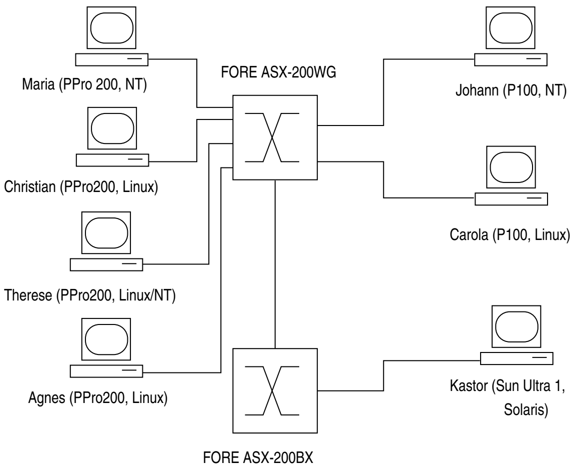

Figure 1: Test environment

2.1 Hardware

The measurements were performed on off-the-shelf Pentium-100 machines[1], Pentium Pro-200 machines[2] and a Sun Ultra 1 model 200 running SunOS (release 5.5.1) as reference machine.

All PC machines included FORE PCA-200E PCI network interface adapters, the Sun used the SBus version of the NIC. The boards support a line rate of 155.52 Mbps and are capable of AAL5 processing in hardware and bus-master DMA transfer.

All machines were physically connected to a FORE ASX-200WG ATM switch using OC-3 optical fiber, the Sun machine was connected via a FORE ASX-200BX switch.

2.2 Software

Some test machines had both Linux and Windows NT installed; others are dedicated Linux and Windows NT systems on identical hardware.

Linux, as a non-commercial monolithic UNIX system originally for i386-based Intel processors, contains full ATM support, including Classical IP over ATM, LAN Emulation v1.0, native ATM support via an ATM API and a flexible device driver interface [2]. The Linux driver for the network interfaces used has been developed in our group and is freely available [3]. For the work presented, the stable kernel 2.0.29, the ATM patch 0.31, and the PCA-200E device driver 0.2 have been employed.

For both Windows NT 4.0 and Solaris the commercial ForeThought software has been used, which includes hardware driver and IP encapsulation.

For our measurements, encapsulation was done according to the Classical IP model [7]. The ATM switch functions as ATMARP server in our environment.

It has been shown that even low-end PC hardware can utilize bandwidth provided by 155.52 Mbps ATM hardware very well[3] [1], as long as the complex state-based TCP protocol is excluded from the data path. This particularly applies if the protocol architecture permits direct copies from applications to the network interface. Due to TCP's required checksum computation reducing data copies is not feasible. Still, Partridge argues that performance improvement techniques for TCP allow seamless integration into a gigabit environment, reducing the common case TCP/IP processing to as little as 150 instructions [9].

We were curious whether TCP actually is a "killer" for high-bandwidth applications on state-of-the-art

[1] 256KB Cache 32MByte RAM, ASUS main boards PCI/I-P54NP4D

[2] 256KB Cache, 64MByte RAM, ASUS main boards P/I-XP6NP5

[3] Using unidirectional transfer with a "single-copy" optimization, the theoretical limit has been approached even for Pentium 90 machines equipped with less powerful boards based on the Intel Neptune chip set.

PC machines. Encapsulation and protocol overhead limits the achievable bandwidth for applications using TCP to 134.5 Mbps for a maximum transmission unit (MTU) size of 9180 bytes [4]. This is shown in Figure 2.

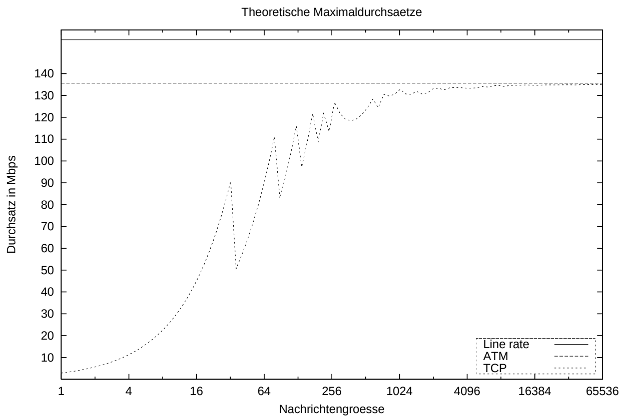

Figure 2: Theoretical achievable TCP bandwidth depending on message size.

For measurements, the `netperf` utility has been used [6]. No particular optimizations involving kernel changes have been made, thus making the results more useful in practice. Only user-level-adjustable parameters, such as message size and socket options, have been varied. During all measurements the participating machines were otherwise unloaded.

3 Expectations

A number of performance problems regarding the usage of TCP on high-speed links are well known [11]. Inadequate socket buffer sizes and inaccurate round trip time estimation may even lead to temporary deadlocks [8]. Protocol processing involving TCP is CPU-intensive, even if data copying overhead is minimized by modern network adapters capable of DMA transfer (as in our case).

From measuring throughput over the loop-back interface[4], an approximate bound could be placed on achievable throughput. Note that the ATM driver is not involved when using the loop-back interface, IP packets re-enter the IP layer via the loop-back driver. Therefore, loop-back numbers hint at machine performance regarding TCP/IP processing. For example, a Pentium-100 running Linux achieved a loop-back throughput of 84 Mbps, thus already practically precluding the possibility of achieving the theoretical maximum TCP throughput for this machine. For comparison of the results, we refer to the netperf results database [5]: For instance, TCP loop-back results for P-100 machines running Win95 indicate that the surprisingly bad number for Johann is no error.

Machine	Operating System	Throughput (in Mbps)
Sun (Kastor)	SunOS	408.88
PPro-200 (Christian)	Linux	347.25
P-100 (Carola)	Linux	84.42
PPro-200 (Maria)	WinNT	338.85
P-100 (Johann)	WinNT	14.23

Table 1: *Loop-back device throughput*

Socket buffer size[5] has a direct influence on the granularity of `read()` and `write()` system calls and the advertised TCP window size. Assuming a round trip time of $1ms$ and an available bandwidth of $135Mbps$, the bandwidth-delay product is $16.8KB$, giving a lower bound on the required socket sizes. Furthermore, it is recommendable that the socket receive buffer is an even multiple of TCP's maximum segment size (MSS[6]) and should hold at least three TCP segments [10]. TCP implementations may raise the socket buffer size to the next even multiple of the MSS size. Therefore, we could reasonably expect to see good performance starting with 32KB socket buffers.

In addition, larger-sized messages should also outperform smaller-sized messages: The reason con-

[4] `netperf -l 60 -H localhost -t TCP_STREAM -i 10,2 -I 99,3 -- -m 65536 -s 65536 -S 65536`

[5] `SO_RCVBUF` and `SO_SNDBUF` are the generic socket options which allow changing the default socket buffer size, which is 64KByte for Linux TCP sockets. The maximum socket buffer size in Linux is 128KB.

[6] For IPv4, $MSS = MTU - 40Bytes$

sists in the reduced number of system calls, lower per-message protocol overhead, and lower TCP processing overhead. By default, TCP does not send small messages immediately if there is outstanding unacknowledged data, rather it assembles segments smaller than the MSS into larger segments to improve throughput (Nagle's algorithm—the socket option `TCP_NODELAY` disables this behaviour.).

For high performance networks the maximum size of the TCP sliding window can severely restrict performance. However, in the LAN environment, the bandwidth-delay product is still small. Linux supports the window scaling option, potentially allowing windows of size 2^{30} bytes.

The hardware purposely chosen for the experiment lets us expect huge performance differences for P-100 and PPro-200 class machines. (The PPro machines are roughly 3 times as fast as P-100 machines, considering integer performance.) The Sun machine was expected to perform about as well as the PPro-200 machines; in addition, the memory subsystem of the Sun is faster than the PC architecture. We did expect the CPU speed to cause performance differences, while we assumed the I/O bandwidth of the PCI bus (60-70MBps) to be sufficient.

4 TCP Throughput

TCP bulk data throughput was measured in the following way: `netperf -l 60 -H 141.76.12.44 -t TCP_STREAM -i 10,2 -I 99,3 -- -m 8192 -s 65536 -S 65536`. That is, `netperf` repeats measurements until they obey the defined confidence interval (which is 97.5%-100% in this case). The influence of message size (1...65536 bytes) and socket buffer size (32KB and 64KB) was evaluated.

Socket Buffer Size. As expected, a larger socket buffer always had a non-negative influence on throughput. Linux did not adapt the socket buffer sizes to the next higher multiple of the MSS which we interpret as a possible reason for Linux profiting more from bigger socket buffer sizes than Windows NT did (see Figure 3). For the remaining throughput numbers, the default $64KB$ socket buffers are used.

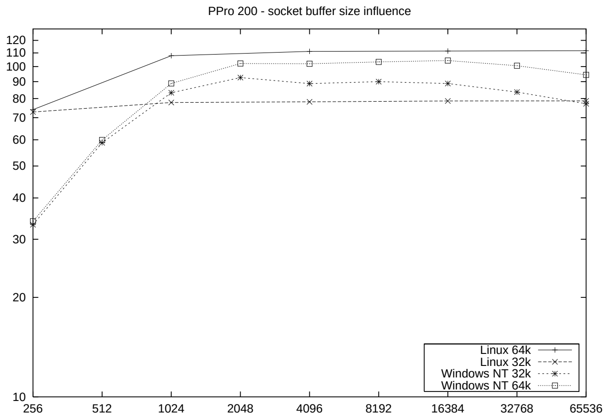

Figure 3: Throughput for 32KByte and 64KByte socket buffer sizes on PPro-200 machines for Linux and Windows NT (in Mbps).

TCP Send Performance. This paragraph presents results for send performance of both the "fast" and "slow" class of machines for WinNT and Linux. As mentioned in Section 3, small messages challenge primarily the protocol processing within the sender. In contrast, for larger messages, receiver performance becomes critical. Figure 4 sketches data path and required copies for `write()` calls. Application data is copied into kernel memory. Linux uses—in contrast to BSD's `mbufs`—fast linear buffers (`sk_buffs`). In-kernel buffers are then copied by the network interface card into the internal send FIFO and transmitted. No CPU involvement is necessary for the last data copy (DMA). Thus, `write()` calls are possible with a single CPU initiated copy.

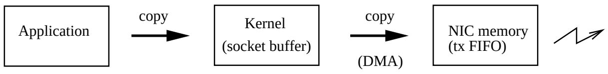

Figure 4: Write data path

As was estimated from the discouraging loop-back performance (Section 3), we found send performance on "slow" Pentium-100 hardware exceedingly better on the Linux machine vs. the WinNT machine (Figure 5). For medium sized messages (e.g., 1024 bytes) a more than three times higher throughput under

Linux was observed, the ratio of peak performances gives $\frac{112.12}{60.99} = 1.84$.

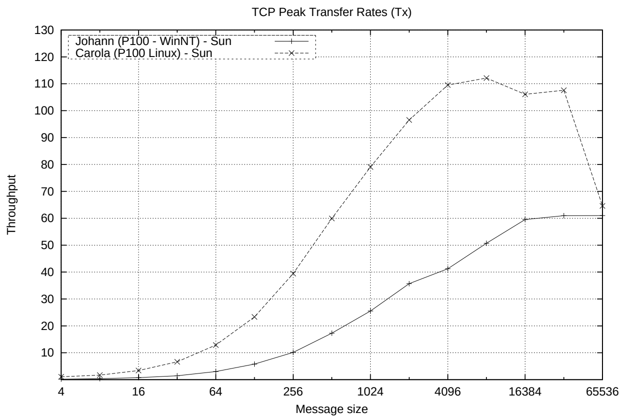

Figure 5: TCP throughput for *slow* senders.

We repeated the previous scenario with more powerful PPro-200 sender machines. Both under WinNT and Linux the theoretical maximum bandwidth has been approached relatively close (up to 88.5 percent). Figure 6 gives the graph.

Similar to the previous scenario, raising the message size to 32KB and 64KB results in lower throughput which particularly affected the Linux machine. We explain this effect by the sender overrunning the receiver's NIC buffers with large messages, causing cell loss and subsequent PDU and packet loss.

TCP Receive Performance. As can be seen from Figure 7, `read()` calls require an extra data copy. Typically, a receive interrupt activity has to identify the higher-level protocol of a PDU, allocate kernel memory, copy the PDU into the socket buffer and return the old buffer to the ATM device driver. In addition, in-kernel control flow is divided into the interrupt activity and the `read()` activity initiated by the receiving application, which blocks until data arrives.

As in the previous paragraph, we isolated TCP receive performance by selecting the Sun machine as sender. The performance graph of both the slow WinNT and the slow Linux machine is given in Figure

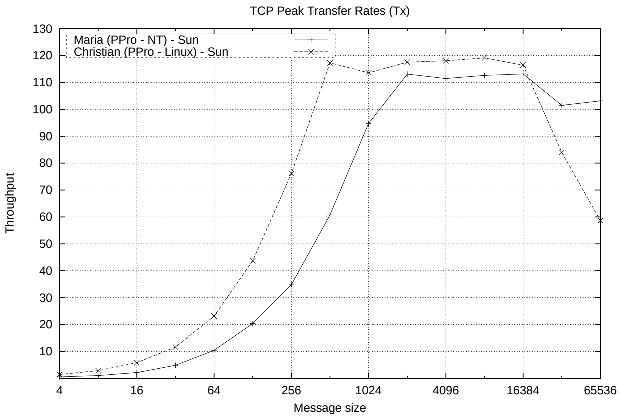

Figure 6: TCP throughput for *fast* senders.

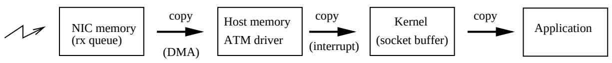

Figure 7: Read data path

8, showing a performance difference of 49% in Linux' favour. Only less than half (Linux) or less than a third (WinNT) of the available bandwidth could be used.

As a final throughput experiment, receive performance of the PPro machines was measured (Figure 9). We got almost identical results for both machines, each utilizing about 75% of the theoretical bandwidth. Note that, recalling Figure 3, the Linux machine can receive at still higher data rates. Therefore, the last scenario hints at the Sun's send performance as limiting factor.

TCP Peak Performance Summary. This paragraph combines the observed peak performances. Contrary to the naive expectations, we found the optimum message size to be in the [4096 . . . 16384] bytes interval. That is, for those messages sizes no throughput anomalies—such as lower throughput caused by receiver overruns—were observed. The achieved peak transfer rates are given in tables 2 and 3, each including performance numbers for communication with the reference machine (Sun).

When comparing the numbers for Linux and Win-

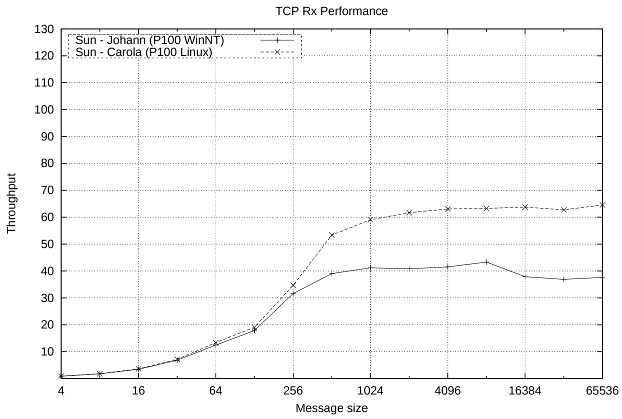

Figure 8: TCP throughput for *slow* receivers.

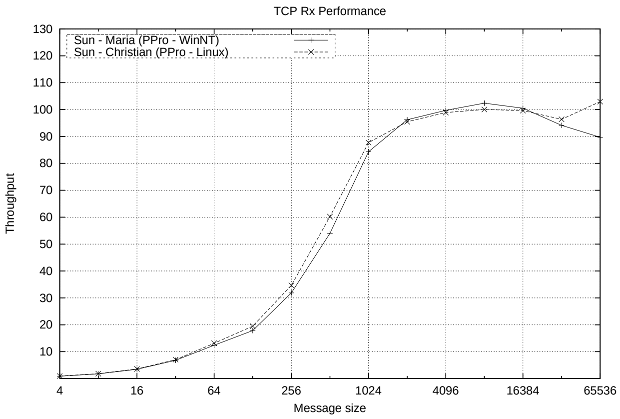

Figure 9: TCP throughput for *fast* receivers.

Linux			
	Receiver		
Sender	P100	PPro200	Sun
P100	-	98.84	112.12
PPro200	33.80	111.65	119.12
Sun	64.57	102.95	-

Table 2: Linux TCP peak transfer rates (in Mbps) for 64KB-sized socket buffers.

Windows NT			
	Receiver		
Sender	P100	PPro200	Sun
P100	-	59.20	60.99
PPro200	38.40	104.29	113.17
Sun	43.31	102.38	-

Table 3: Windows NT TCP peak transfer rates (in Mbps) for 64KB-sized socket buffers.

dows NT, Linux outperformed Windows NT particularly on slow (P-100) hardware. If running on fast hardware, throughput relatively close to the theoretical maximum value of 134.5 Mbps can be achieved. For small messages, protocol and system call overhead dominates and results in a high load at the sender. For large messages, the memory subsystem of the receiver cannot yet preserve the throughput and routinely loads the CPU fully.

5 TCP Request–Response Performance

For request–response protocols measurement of the TCP round trip time is most significant. The following command line gives an example for the measurement technique used: `netperf -l 60 -H 141.76.12.46 -t TCP_RR -i 10,3 -I 99,5 -- -r 64,64 -s 0 -S 0`. There was no need to explicitly disable TCP's Nagle algorithm since such ping-pong communication is not affected. The varying size of request and response data—taken from `netperf`'s default test suite for request–response tests—reflects typical client-server communication (table 4).

Firstly, as a typical scenario communication of slow clients (Pentium-100) with fast servers (PPro-200) is examined. For comparison, the graph for communication of the same client with a different server (the Sun machine—Kastor) is being given in Figure 10.

All request–response measurements resulted in Linux consistently outperforming Windows NT by a large margin.

Secondly, Figure 11 shows achieved overall peak performances, using PPro-200 computers for both

Request size	Response size
1	1
64	64
100	200
128	8192

Table 4: Request and response message sizes (in bytes)

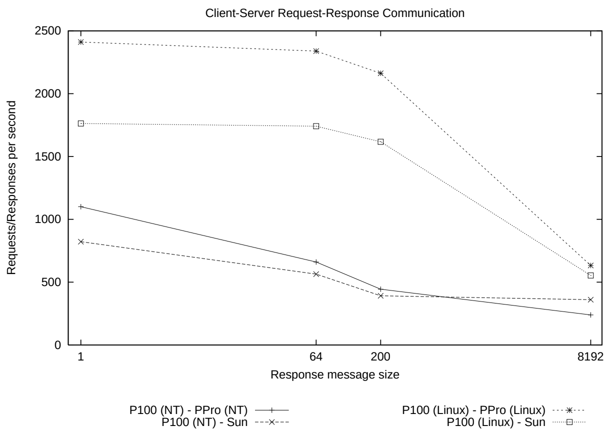

Figure 10: Number of request–responses per second for client-server communication.

Windows NT and Linux. For Linux, a performance of 3387.43 requests and responses per second has been observed, corresponding to an application-to-application round trip time of $\frac{1s}{3387.43} = 0.295ms$. On the same hardware, Windows NT achieved a round trip time of $\frac{1s}{2250.87} = 0.444ms$. For a more realistic request size (128 bytes) and response size (8192 bytes), round trip time increased to 0.964 ms (Linux) and 1.372 ms (Windows NT). This gives an speed advantage of $43.7\% \ldots 50.5\%$ for Linux, comparing request–response performance on identical PPro-200 machines.

It is worth stating that additional measurements with "slow" P-100 servers running Linux offered better response times than both a PPro-200 running Windows NT and the Sun Ultra (answering null-RPCs).

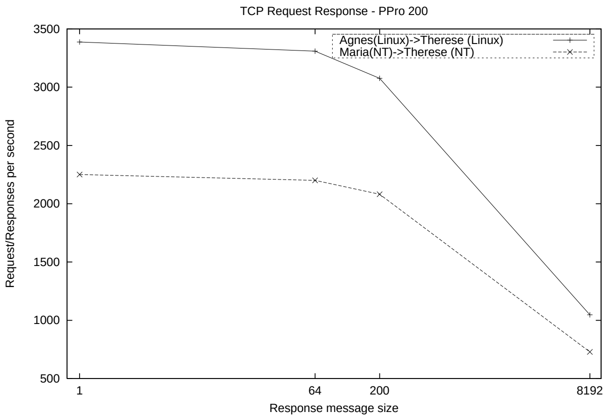

Figure 11: PPro-200 WinNT vs. Linux: Number of requests and responses.

6 Conclusion

We compared achievable application-level performance of TCP over ATM for two different operating systems. We did not interpret all numbers in detail, some have to be taken "as is". This is partly due to missing insight into the WinNT protocol architecture, and to a lesser extent into the SunOS protocol implementation. Nevertheless, some observations can be made.

State of the art PC hardware can utilize available bandwidth over ATM well, even when running a complex transport protocol on top of ATM. However, we found that CPU load on the participating machines is very high. It can be concluded that host performance is still relatively low in comparison to the bandwidth offered by the network. In other words, there is no urgency to replace the 155.52 Mbps ATM technology in the local environment by faster technologies.

When comparing protocol stack and driver performance of Windows NT and Linux, a number of points can be raised:

1. Performance achievable at application level mainly depends on CPU processing power and efficient implementation of the TCP/IP stack.

2. TCP throughput is in the same ballpark for both Linux and Windows NT using fast ma-

chines. However, for slower hardware (e.g., Pentium 100) there is a clearly visible performance advantage for Linux.

3. From an application point of view, most consistent throughput results have been achieved with message sizes between 4KByte and 16KByte.

4. TCP request–response performance is much higher for Linux, yielding better round trip times in the order of 50%.

5. Reviewing the assumptions made in Section 3, the request–response measurements proved that Linux is able to achieve round trip times significantly smaller than 1ms. It follows that even in the local environment the bandwidth-delay product is high enough to make traditional TCP window sizes (i.e., less than 64KB) potential bottlenecks.

Useful numbers we left out in this presentation include the exact CPU utilization of the machines. This is owed to the lack of applicable quantitative measurement tools on the platforms examined.

7 Acknowledgements

The author wishes to thank Robert Baumgartl for his comments, and Sven Rudolph for supplying configuration details and general insights.

References

[1] Werner Almesberger. High-Speed ATM networking on low-end Computer Systems. Technical report, LRC Lausanne, 1995.

[2] Werner Almesberger. ATM on Linux. http://lrcwww.epfl.ch/linux-atm/, 1997.

[3] Martin Borriss and Uwe Dannowski. Linux Support for FORE Systems PCA-200E NIC. http://os.inf.tu-dresden.de/project/atm/, 1997.

[4] John David Cavanaugh. Protocol Overhead in IP/ATM Networks. Technical report, Minnesota Supercomputer Center,Inc., 1994.

[5] Rick Jones. The Netperf Results Database. available from http://www.cup.hp.com/netperf/numbers/NetperfBrowse.html.

[6] Rick Jones. The Public Netperf Homepage. http://www.cup.hp.com/netperf/, 1997.

[7] M.Laubach. Classical IP and ARP over ATM. RFC 1577, 1994.

[8] Kjersti Moldeklev and Per Gunningberg. Deadlock situations in TCP over ATM. In *IFIP Workshop on Protocols for high speed networks*, August 1994.

[9] Craig Partridge. *Gigabit Networking*. Addison Wesley, 1994.

[10] W. Richard Stevens. *Unix Network Programming*. Prentice-Hall, 2nd edition, 1998.

[11] V.Jacobson, R.Braden, and D.Borman. TCP Extensions for High Performance. RFC 1323, 1992.