33

Feasibility of a Software-based ATM cell-level scheduler with advanced shaping

J. Schiller
Institute of Telematics
University of Karlsruhe
Karlsruhe, Germany
j.schiller@ieee.org

P. Gunningberg
Department of Computer Systems
Uppsala University
Uppsala, Sweden
per.gunningberg@docs.uu.se

Abstract

Future servers are expected to handle a large number of connections with different Quality of Service (QoS) requirements. Networks, e.g., based on ATM technology, provide QoS via standardized traffic parameters. While the control at the network edge can handle these parameters, support for generating adequate traffic patterns (shaping) in the end-systems is limited due to hardware restrictions. This paper presents a software-based cell-level scheduler for ATM with advanced shaping mechanisms supporting priorities and fair sharing in overload situations. Furthermore, a scalable integration of the cell-level scheduling and the scheduling of DMA transfers is shown.

Keywords

ATM, shaping, scheduling, DMA, network adapter, proportional sharing, leaky bucket

1 INTRODUCTION

Future network servers are expected to handle a large number of concurrent connections with varying Quality of Service (QoS) requirements (Campbell, 1996). Examples of likely traffic sources are voice/video conferences integrated with multimedia documents, multimedia document retrieval, WWW, file transfer and real time interactive games. The bandwidth requirements will range from a few Kbit/s to

Broadband Communications P. Kühn & R. Ulrich (Eds.)

several Mbit/s. Thus, servers using network technologies with link bandwidths of hundreds of Mbit/s have the potential to carry a substantial number of these connections. The problem which arises is to handle all these connections and their QoS requirements. Current WWW servers serve requests using a best-effort method, which is not sufficient when the individual connections have QoS requirements and as the number of connections increases. We assume that a future large WWW server must handle many concurrent connections with potentially highly different QoS requirements.

The network adapter of such a server is responsible for several functions in the outbound direction. It should transfer data from server memory, apply low level protocol functions, control the transmission of data according to a traffic contract and multiplex several connections. Furthermore, in our view a more advanced network adapter is needed to perform traffic shaping. Our thesis is that a CPU based approach for doing these functions for a large number of connections is both feasible and efficient. Using software and state of the art microprocessors has several benefits. The algorithms can easily be changed to suit the needs of different traffic types. In addition, a software solution benefits of new on-board processors as they become available. Our solution will scale with the increase in processing power provided the memory system keeps up. An additional issue which is investigated in this paper is if and how the traffic shaping and the scheduling of memory transactions can be combined.

The contribution of this work is a feasibility study of a software scheduler running on an on-board network adapter CPU combining the traffic shaping functionality with the scheduling of memory transactions. These operations are often decoupled, leading to hardware redundancy, double buffering of data, and the limited capacity for shaping in the I/O subsystem.

The network technology we have envisioned is ATM since it supports QoS parameters and traffic contracts for every connection. However, the proposed traffic shaping algorithms used by the network adapter are general and may be employed in any network technology with flow reservations, such as the next generation Internet protocols. In this work, we focus on how to schedule connections which have been accepted by the system. The connection acceptance phase and the QoS architecture for the applications and higher layer protocols (Campbell, 1996) which are also crucial to the server is outside the scope of this work.

2 EXISTING HARDWARE SOLUTIONS AND THEIR LIMITATIONS

Comparing our approach with the capabilities of actual shaper-chips the following topics have to be addressed:

- *Scalability*: how does the approach scale in the number of connections, where are restrictions introduced by hardware, data structures etc.
- *Efficiency*: how efficiently can the shaper utilize the outgoing bandwidth.
- *Overhead*: how large is the overhead of the implementation, e.g., how often has a host system to be interrupted, how much control information has to be exchanged between host memory and the adapter.
- *Isolation*: how well are different connections protected from each other, what happens in transient overload situations, and how individual parameters can be adjusted to the needs of a connection.

The scalability of all chip-based solutions is very limited. Values like the maximum number of supported connections (VC) are of more theoretical nature and typically only limited by the width of registers. More interesting is the number of *simultaneously* supported connections or the number of connections supported *on-chip*. These numbers of supported connections rely not only on the amount of memory to store connection state date, but rather on the algorithms, data structures, and available processing power for traversing connection information.

There are typically limits on the maximum number of *different* traffic characteristics. Additionally, the question arises what parameters can be set and if they can be used independently. Typical restrictions are 8-12 PCR (Peak Cell Rate) queues, every connection has to be in one of these queues (Fujitsu, 1997), (SIEMENS, 1997). Furthermore, the SCR (Sustainable Cell Rate) is often derived from the PCR via a per connection ratio (e.g., ½ or ¼ of the PCR). These restrictions are due to the fact, that the cell rates are generated using explicit hardware counters, only a very limited number of these fit on a chip (c.f. (ATM Forum, 1996) for further explanation of ATM traffic parameters).

Also due to space limitations, most of the data structures holding context information for a connection are located in host memory. To update these structures the host system has to be interrupted and the memory accessed. This can result in a large overhead, especially when these data structures have to be checked with full PCR (Fujitsu, 1997). Another topic is the DMA overhead, most of the approaches transfer single cells from host memory 'just-in-time' due to very small transfer buffers on chip. Typically, the receiving side has a higher priority to avoid cell losses. Thus, the sending side suffers if cells arrive or even if control information has to be updated (SIEMENS, 1997).

The isolation between different connections sharing one PCR queue is typically not addressed in any of the evaluated solutions. Some solutions provide additionally priority classes, the behavior within a priority class during transient overloads is not further determined.

2.1 UPC solutions

It is important to compare the capabilities of the UPC (Usage Parameter Control) chips located at the UNI (User Network Interface) with those of the shapers due to the fact that the UPC chips will decide if an incoming cell of a connection shaped by one of the shapers is accepted or not. The first fact one notices is the more detailed control capabilities and the variety of parameters to check. The ATM_POL3 from ATecoM (Atecom, 1997) can control up to 64k VCs checking PCR, SCR, CDV (Cell Delay Variation) and maximum burst size using a dual leaky bucket per VC. IgT has the WAC-186-B (Integrated, 1995) which uses GCRA to monitor PCR, SCR, CDV and burst tolerance for up to 16k active VCs for a bitrate up to 250 Mbit/s. Finally, the BNP2010 UPC of National Semiconductor (National, 1997) checks up to 16k VC with 3 GCRAs per VC up to a speed of 622 Mbit/s. A simple comparison shows, that the UPC can be much more precise and, hence, more restrictive than the capabilities of the best shaper chips.

3 ARCHITECTURES

The overall architectural design is illustrated in Figure 1. The adapter has a buffer memory to hold cells to be transmitted and a DMA engine that reads data from server

memory into the buffer memory. Each connection has a separate queue of cells in the memory. Buffer memory is accessed from the ATM chip which does the actual transmission. The purpose of the CPU is to schedule the DMA engine for a data transfer, schedule the ATM chip to read a cell from the buffer memory according to the traffic shape and multiplexing state, and exchange control messages with the host. Note that the CPU does not touch data at all since this would be too time consuming.

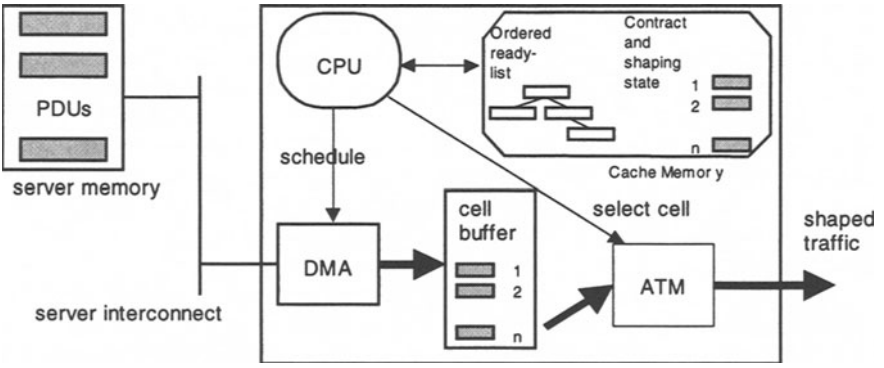

Figure 1 Network adapter architecture.

The CPU is running the following cycle for each transmitted cell.

1. Pick the first connection identifier in the ordered ready-list of active connections, provided it is due for transmission.
2. Initiate the ATM chip for a transfer of a cell from this connection.
3. Shape the connection and calculate the time when the next cell of this connection should be sent according to the shaping state.
4. Insert the connection identifier again at a new place in the ready-list according to the next transmission time.
5. Schedule a possible DMA transfer of data from server memory for this connection.

This cycle must finish well within the time it takes to send a cell. For a 622 Mbit/s this means within 680 ns. Besides running this cycle, the CPU needs to synchronize PDU information with the server and to allocate and deallocate buffer memory. But these tasks are triggered by asynchronous events and can be done in the background.

The CPU needs a fairly large memory to hold the state of each connection and the data structure for the ready-list. The access time to this memory is crucial for the performance and it is expected that a large Level 2 (L2) cache memory is the most appropriate. The actual scheduling code is small enough to fit into Level 1 (L1) instruction cache. The adapter may have other protocol hardware on board, such as for AAL5 checksum calculation which also must be controlled by the CPU. In addition, there is some control logic that is specific to the interconnect.

4 INTEGRATED SCHEDULING, ADVANCED SHAPING, AND DMA TRANSFER

The novelty of this design is the way shaper, scheduler, and DMA transfer cooperate to fulfill the task of sending the right cell at the right time according to traffic contracts

and the actual load of the adapter. The design idea is that the CPU co-schedules both the DMA and ATM functions. The CPU has enough information to bring in data from the server memory just in time for transmission since it is deterministic when the next cell in a connection is allowed to be sent according to the shaping algorithm. Furthermore, the CPU is also in full control of the multiplexing of several cells by maintaining an ordered ready-list of connections ready for transmission. With this information it is predictable when a cell will be transmitted and hence the latest time when data must be fetched from server memory. The focal point is the traffic shaper. The shaper holds the context information of a connection, i.e., PCR (Peak Cell Rate), SCR (Sustainable Cell Rate), CDVT (Cell Delay Variation Tolerance), burst size, amount of data already sent etc. The scheduler hands over a pointer to the next connection to be shaped and the shaper feeds the scheduler with timing information about the earliest time a cell from the connection can be transmitted (Figure 2). The shaper state holds information about the actual amount of data for a connection on the adapter and the PDUs stored in host memory. By interpreting this state, new data can be pre-fetched from server memory when it is necessary and viable.

4.1 The scheduler

The purpose of the scheduler is to determine when a connection is allowed to send. A connection is assigned a cell slot for transmission sometime in the future, depending on the connection state and the contention of slots between connections due to multiplexing. The scheduler maintains an ordered data structure of connections, ordered with the connection to send closest in time first and the latest at the end. The scheduler gets the earliest possible time a connection can be scheduled from the shaper. It will then try to allocate the corresponding cell slot time by checking the ready-list. If this slot is already occupied by another connection the scheduler may try to find an empty slot later in time or to reschedule the conflicting cells using the CDVT (Cell Delay Variation Tolerance) parameter or to use a static priority. An earlier slot time cannot be used by the scheduler, even if there are several empty slots since the cell will then break the traffic parameters. Such a cell would most likely be discarded by the UPC (Usage Parameter Control) mechanism as an early cell outside the contract. Scheduling later is always acceptable by UPC, but may affect the end-to-end guarantees and results in less bandwidth efficiency. Design issues for the scheduler include efficient utilization of the bandwidth, fairness at overload situations and the minimizing of the number of CPU cycles needed for the scheduling. Our fairness proposal and our measurements on CPU cycles will be discussed later.

4.2 The ready-list data structure

The ready-list has one entry for each active connection. For 64K connections, the size of this list is considerable and the time it takes to keep the list sorted is critical for the performance. The data structure chosen for this task is a binary tree, implemented as a heap. A heap can realize a binary tree efficiently without the use of pointers. All information kept in the heap has to be small to assure that most of the heap fits into the cache of a processor (e.g. 512k L2 cache for a PentiumPro). The connection state has to be fetched into the cache only when a connection is scheduled. This structure allows for a large number of connections handled simultaneously. Assuming 32 bit values for the

cell slot time and a pointer to the state the heap needs only 512kbyte for 64k active connections.

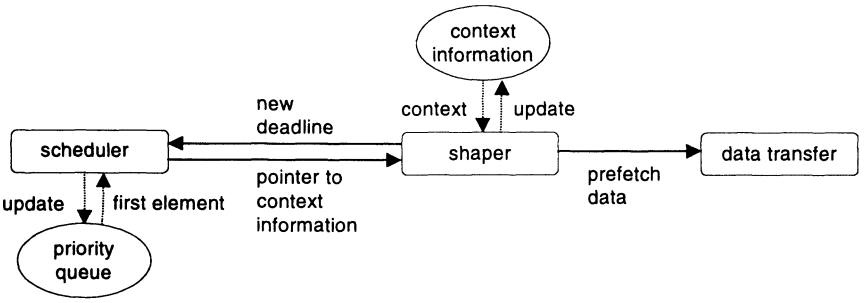

Figure 2 Interaction of shaper, scheduler, and data transfer engine.

4.3 Shaper

When called with a connection identifier, the shaper updates the shaping state and calculates the earliest time the next cell of the connection can be scheduled without violating the traffic contract.

The implementation of the traffic shaper uses a combination of the VSA (Virtual Scheduling Algorithm) and LBA (Leaky Bucket Algorithm), which we derived from the LBA and VSA specifications for the GCRA (Generic Cell Rate Algorithm) in the ATM UNI (ATM Forum, 1996). Instead of using these algorithms for controlling conforming cells at the UNI, we actively shape the traffic using them. This guarantees that all cells shaped with our implementation will be accepted by the UPC. The calculation is based on the current state of the connection (PCR, SCR, inactive) and the mode (single/dual leaky bucket, SLB/DLB). If the connection is in PCR or SCR, the shaper returns the new earliest transmission time for the next cell. Thus, it is guaranteed for the SCR state that a new token will be available if this connection is scheduled the next time. If a connection uses the DLB mode and no more tokens are left, the shaper returns the time when the whole token bucket can be refilled completely with tokens. Note that the shaping state of a connection will be accessed *if and only if* a cell can be scheduled. There is no other updating necessary, e.g., filling new tokens in the bucket. Many current implementations have to access state permanently to update the schedule resulting in a poor performance and very limited number of connections handled at the same time (Fujitsu, 1997), (LSI, 1997), (SIEMENS, 1997).

4.4 DMA transfer from server memory

The connection state has information about the amount of currently stored cells on the adapter and a list of current PDUs stored in the server memory. Given the deterministic information from the traffic shaping it is possible to keep most of the PDUs in server memory and to move data just in time for transmission to the adapter. If all PDUs have been sent the adapter notifies the server and the connection will enter a non-active state. There are several potential advantages by using this deterministic information: pre-fetching of the right amount of data will avoid delays caused by demand fetching, buffer size requirements can be reduced and long blocking times to other interconnnect transactions can be avoided. By coupling the DMA transfer for a connection with the

actual sending of cells for this connection the design is simplified. The amount of cell buffer memory needed can then be decided and the shared buffer problem with asynchronous readers and writers is avoided, since the transfer is synchronized with the transmission, i.e., the consumption of data.

Two design issues must be addressed. The first one is *when* a transfer should take place. The interconnect has some access time variation that must be compensated for. This variance motivates an earlier transfer than just before the data is needed. The second issue is the *size* of the data transfer unit. The smallest unit is a cell and the largest is the PDU. A small size will cause more overhead while a big unit will consume cell memory buffers and may block other transfers. In this trade-off, the optimal transfer size of the interconnect must also be considered for efficiency.

Figure 3 shows a small example schedule generated by our prototype. 10 connections with different shapes, starting points and number of cells are scheduled together. The y-axis shows the consumption of cells, the x-axis the number of the cell slot the cell is scheduled in. Connections 2, 3, 4, 7, 8, and 9 are CBR connections, the other VBR. Connection 5 uses SLB, connections 0, 1, and 6 DLB mode. The exact shape of the plotting lines depends not only on the connections parameters but also on the current state of the other connections. This results, e.g., in slight deviations from an ideal straight line when two or more connections are scheduled for the same cell slot.

4.5 Priorities and Proportional Sharing

As soon as one implements an algorithm for scheduling one very important question is how the implementation behaves in overload situations. Overload situations can occur quite frequently, if one does not want to make only conservative reservations, i.e., allowing the sum of all PCRs never to be greater than the total capacity of the link. This would result in a very poor overall utilization if, e.g., VBR traffic sources are used. Here the PCR can be easily 10-1000 times larger than the SCR. One example is the transfer of MPEG2 coded video streams. Typical values are 1.0 to 15.0 Mbit/s for PCR, 0.2 to 4.0 Mbit/s for SCR. In addition, quite often a priority scheme is required to weight different traffic streams. One could for example give voice connections a higher priority as connections to fetch pictures from Web-pages. This would result in a higher audio quality and only minimal additional delay for the picture data transfer.

Assuming this, one can refine the question concerning the overload situation into following questions:

- *Sharing*: How is the available bandwidth shared within one priority class? What happens in overload situations caused within a priority class?
- *Isolation*: How is the interaction between different priority classes? What happens if a higher priority class already causes an overload?
- *Stability*: What happens to the system if the overload situation continues for a longer time? Does the system still provide a schedule "as good as possible"? Is the system stable?

From a users point of view the first set of questions can be answered as follows. If a user has started, e.g., several video applications that load the network completely and now starts an additional one he or she expects the available bandwidth to be shared fairly between the applications. The communication system can not make any assumptions of the importance of an application, and therefore a proportional sharing

scheme is the best one can do. That means, that an application that used twice the bandwidth compared to another one still gets twice as much as the other one. But now this is less than before due to the overload. To privilege one application, one can shift the application to a higher priority. Our implementation guarantees this proportional share within one priority class independently for every connection.

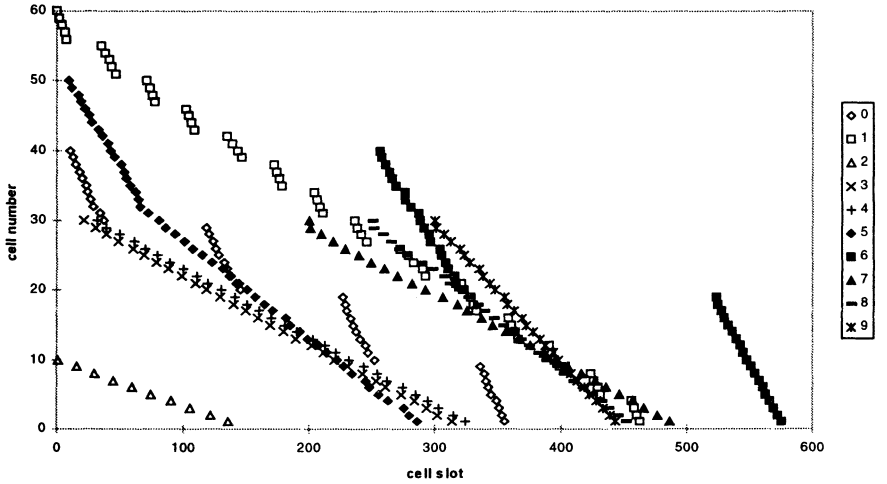

Figure 3 Example cell level schedule.

Prioritizing an application leads to the second set of questions concerning the interference between priority classes. Depending on the scheduling policy one can decide for a hard priority scheme, i.e., the scheduler tries first to satisfy connections with higher priorities and ignores lower priorities in case of an overload. This results in starvation of connections in lower priority classes. An alternative solution could provide a minimum share of the total bandwidth to avoid starvation. This is in general the better alternative due to the fact that overload situations are typically transient and common communication protocols like, e.g., TCP cannot deal properly with a total starvation but adapt well to lower bandwidth. Our implementation allows both alternatives by guaranteeing proportions of the total bandwidth in an overload situation.

All proportional sharing and handling of overload is done within the traffic shaper via adapting traffic parameters as soon as a connection is shaped and the load situation changes. This guarantees also that the prefetching of data is always harmonized with the real sending rate, so that no internal buffers can overflow in the communication system. The adaptation of the application to lower bandwidths is out of the scope of this work, but generally, this is the scheme many approaches tend to incorporate. No matter how long an overload situation exists or how strong the overload is, the system will always try to generate a schedule as close as possible to the traffic contract. The settings of the proportional sharing between different priority classes is left to the operator and depends on the policy of a service provider.

Proportional sharing and the overload behavior described above are up to now not implemented in any of the available traffic shaper chips. Either these implementations avoid overload situations by not allowing over allocation (LSI, 1997) or they throttle

the total traffic via a leaky bucket (Fujitsu, 1997) (SIEMENS, 1997). Figure 4 shows an example for the effects of priority classes and proportional sharing within a priority class. Connection VC 3 has the highest priority 0, the connections 1, 4, 5, and 6 are in the same priority class 1, and finally connection 2 has the lowest priority 2. The connections 4, 5, and 6 are configured to require already 100% of the bandwidth per connection. This results in an heavy overload situation between cell slot 200 and 400. Due to the higher priority, connection 3 is not disturbed. The proportional sharing within one priority class can be seen for the connections 1, 4, 5, and 6. The slope of the graph flattens, as soon as a new connection with the same priority starts (at cell slots 200, 250, and 300). This demonstrates the proportional sharing: having the same priority and traffic parameters, two graphs must have the same slope. Finally, connection 2 has the lowest priority and starves during the heavy overload situation. If required, this could be avoided by reducing the bandwidth available for connections with higher priorities as described above.

5 PERFORMANCE EVALUATION

The algorithm was implemented using C and tested on a PentiumPro with 200MHz and 512k L2 cache (Windows NT 4.0 and Linux 2.0.28), a Digital Alpha AXP 3000/800 (Digital UNIX 3.2D-2) and a Sun Ultra I with 143 MHz (Solaris 2.5.1). The implementations on the different machines differ only in the instrumentation, not in the algorithm. The measurements were done running the complete operating system concurrently, but no other application programs. This was done on purpose to see the behavior of the algorithm in a real environment with current operating systems and not on specialized stand alone systems. The presented results give therefore an upper bound for execution times and cycles counts. The performance can only increase on dedicated systems.

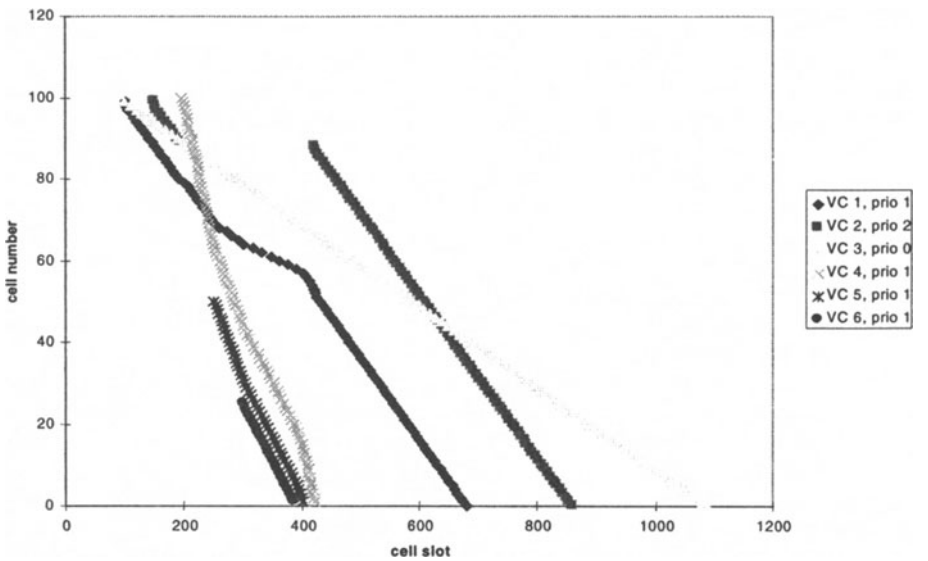

Figure 4 Effects of transient overloads, proportional sharing and priorities.

The main platform for instrumentation was the PentiumPro, Alpha and Ultra SPARC processors were used for comparison. Our main interest is in the number of CPU cycles used for scheduling of one cell. This includes the updating of all data structures, issuing of data transfer commands if necessary, and shaping of the cell stream. For counting of the CPU cycles the time stamp counter (TSC) of the PentiumPro was used. This allows for a resolution of single CPU cycles (5ns for the 200MHz CPU used). The TSC is a free running 64 bit counter not influenced by system events.

To evaluate the performance of the implementation we run worst-case scenarios that load the data structure heavily and require almost always the worst case of updating operations needed for the heap. One such scenario is for example the setup of 64k *simultaneously active* connections with identical parameters. The algorithm implemented puts no restriction on the number of connections, amount of data, or link speed. Only the actual performance of a given CPU/memory system limits this performance. To give an impression of the performance of the implementation also typical configurations were evaluated. It has to be remembered that this implementation treats every connection separately, i.e., no connections are combined or share common properties as this is the case in all existing hardware solutions due to the limited number of registers available.

Figure 5 shows cycle counts on a PentiumPro for more than 64k simultaneous active connections each sending 25000 bytes in 5 PDUs. The bandwidth chosen for the connections does not influence the performance of the algorithm, 9600 bit/s were chosen to result in a reasonable aggregated bandwidth of 616 Mbit/s (e.g., for a 622 Mbit/s adapter). Overloading an adapter using this algorithm, i.e., accepting a higher aggregated bandwidth than the total bandwidth of the adapter, does not result in a performance degradation but in an overall higher delay for cells. This is the best one can expect if the overload is done on purpose and no cells should be dropped. Most of the cells can be shaped and scheduled within 600 CPU cycles. This includes data transfer commands if necessary. Only at some points in time the cycle count goes up to about 650 cycles. It can be shown running the same algorithm on an UltraSparc that the jump in the cycle count at the beginning is a result of the PentiumPro L2 cache and not the algorithm. These UltraSparc measurements took place running a full installation of Solaris 2.5.1 but no other application programs resulting in an average time for shaping, scheduling, and data transfer of 4.5µs. To be able to handle the large size of instrumentation files values were collected and averages calculated. In addition maximum values were controlled to make sure that they are not averaged out. The binsize used for averaging is noted in the figures. 600 CPU cycles represent for the chosen processor a real-time of 3µs (200 MHz).

To stress the implementation all connections are started at exactly the same time, i.e., all data is tried to prefetch for the complete cell buffer at the beginning and then also consequently for every new PDU of a connection. The main result of these measurements is not the single number of cycles used but the fact that the number of cycles has an upper bound even under worst case conditions and has a very stable behavior for most of the cases.

Where the performance results with 64k connections show that with today's processors and standard operating systems cells cannot produced fast enough for, e.g., 155 Mbit/s adapters, this is possible for a lower number of connections. Figure 6 shows that a software solution of a shaper/scheduler can produce a cell in less than 350 cycles, i.e., 1.75 µs. This is definitely fast enough for a 155 Mbit/s adapter. Again this

configuration is a worst case for 1000 connections, the cycle count per cell will drop if for example some high bandwidth connections together with a substantial number of low bandwidth connections have to be scheduled. With this lower number of connections no jump in the cycle count due to the cache behavior can be seen.

Loading the implementation with 100 connections results in a cycle count per cell of typically less than 300. If only one active connection is configured the cycle count drops to 160. The shaping takes less than 85 cycles on average and is independent of the size of the data structure.

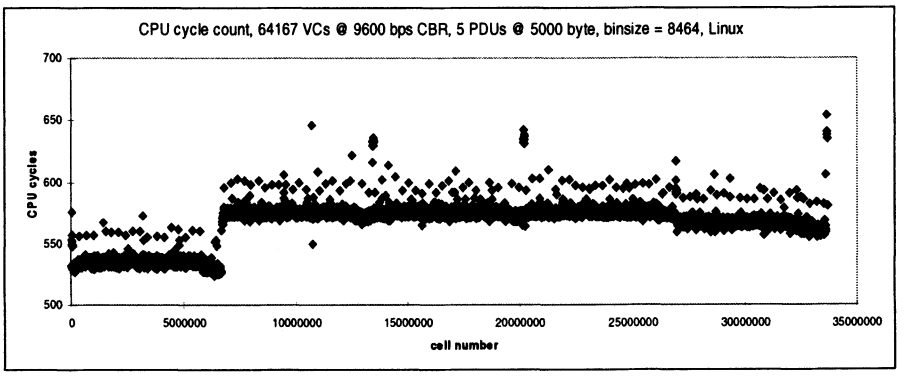

Figure 5 Cycle count on a PentiumPro/Linux sending data for 64k connections.

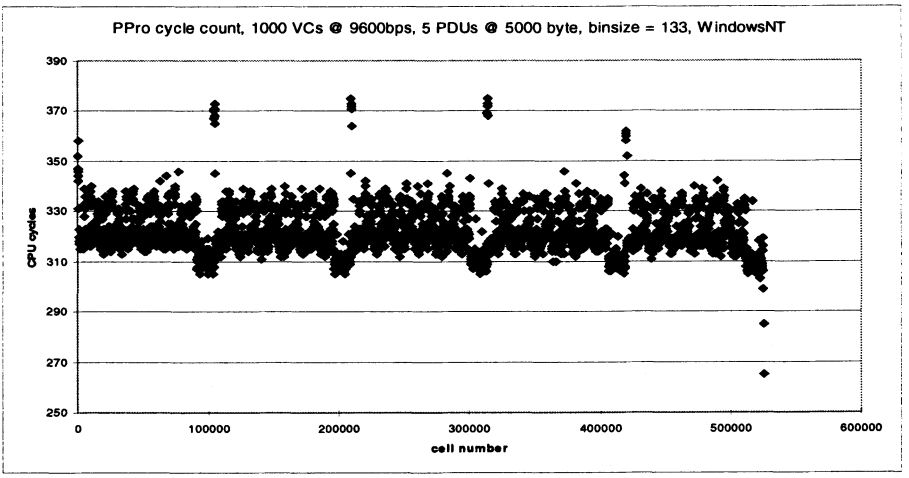

Figure 6 PentiumPro cycle count for 1000 connections.

6 CONCLUSIONS

Today, a discrepancy between the capabilities of shaping solutions, i.e., mostly chips to generate certain traffic patterns, and the UPC chips, e.g., at the UNI exists. This may lead to cell loss, although the traffic parameters agreed upon are the same for the shaper and UPC. One reason for this are the very limited capabilities of hardware solutions for shaping due to a limited chip area. Our work shows, that it is already today feasible to

shape traffic for ATM networks for over 1000 connections at a rate of 155 Mbit/s using a today's general purpose CPU. Furthermore, with the proposed software solution cell-level scheduling and the scheduling of DMA transfers can be harmonized resulting in an overall higher efficiency and lower buffer requirements.

Further work will concentrate on performance measurements using high-end workstations with 600MHz CPUs and the integration of the DMA scheduling into the scheduling mechanisms of the operating system to further harmonize data transfer.

6 REFERENCES

AtecoM (1997) ATM_POL3, http://www.atecom.de/

ATM Forum (1996) Traffic management specification, version 4.0, ATM Forum

Campbell, A., Aurrecoechea, C.: Hauw, L. (1996) A Review of QoS Architectures, Proceedings of the 4. International IFIP Workshop on QoS (IWQoS), Paris

Coulson, G., Campbell, A., Robin, P., Papathomas, M., Blair, G., Sheperd, D. (1995) The design of a QoS-controlled ATM-based communication system in Chorus. *IEEE Journal on Selected Areas in Communications*, **13(4)**, 686-699

Digital Equipment Corporation (1996) Program Analysis Using Atom Tools. Digital Equipment Corporation, Maynard, Massachusetts

Druschel, P., Banga, G. (1996) Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems. Proceedings of the USENIX Association Second Symposium on Operating Systems Design and Implementation, Seattle

Druschel, P., Peterson, L.L., Davie, B.S. (1994) Experiences with a high-speed network adapter: a software perspective, ACM SIGCOMM, London

Dalton, C., Watson, G., Banks, D., Calamvokis, C., Edwards, A., Lumley, J. (1993) Afterburner. *IEEE Network*, pp. 36-43

Engler, D.R., Kaashoek, M.F., O'Toole, J. (1995) Exokernel: an operating system architecture for application-level resource management. ACM SIGOPS

Fujitsu (1997) ALC (MB86687A), http://www.fmi.fujitsu.com

Georgiadis, L.; Guerin, R., Peris, V., Sivarajan, K.N. (1996) Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Trans. Networking*, **4**

Gopalakrishnan, R., Parulkar, G. (1995) A Framework for QoS Guarantees for Multimedia Applications within an Endsystem, 1. Joint Conference of the Gesellschaft für Informatik and the Schweizer Informatikgesellschaft, Zürich

Integrated Telecom Technology (1997) WAC-186-B, http://www.igt.com

LSI Logic (1997) ATMizer II (L64363), http://www.lsilogic.com

National Semiconductor (1997) BNP2010 UPC, http://www.national.com

Rexford, J., Bonomi, F, Greenberg, A., Wong, A (1997) A Scalable Architecture for Fair Leaky-Bucket Shaping, IEEE Infocom, pp. 1056-1064

SIEMENS (1997) SARE (PBX4110), http://www.siemens.de

Toshiba (1997) Meteor (TC35856F), http://www.toshiba.com

TranSwitch (1997) SARA II (TXC-05551), http://www.txc.com

Traw, C.B.S., Smith, J.M. (1993) Hardware/software organization of a high-performance ATM host interface, *IEEE JSAC*, **11(2)**, 240-253

Wrege, D.E, Liebeherr, J. (1997) A Near-Optimal Packet Scheduler for QoS networks. IEEE Infocom, Kobe