Feedback controlled scheduling for QoS in communication systems

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

Abstract

Many traditional mechanisms for QoS (Quality of Service) provisioning failed due to complexity, incompatibility to existing applications, or the need for new computer architectures. To be successful, an approach should be integrated into commercial widespread systems, simple, and controllable. This paper discusses basic problems of traditional approaches and proposes a communication system based on feedback controlled schedulers. This enables automatic adaptation to system parameters and changes in the environment. One crucial component for QoS in communication, a scheduler for data packets, will be discussed in more detail. The performance evaluation shows, that it is feasible with the proposed software approach to reach high throughput and to guarantee a broad range of QoS parameters for individual traffic flows.

Keywords

QoS, shaping, scheduling, network adapter, feedback mechanisms, adaptive applications

1 INTRODUCTION

One basic question, that motivated many research activities, is the provisioning of (guaranteed or statistical) QoS for communication oriented applications by the underlying communication subsystem (Campbell, 1996), (Aurrecoechea, 1998). The motivation for QoS is based on the demands of multiple multimedia applications communicating over a large range of communication media (radio,

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35388-3_42

H. R. van As (ed.), *High Performance Networking*
© IFIP International Federation for Information Processing 1998

fibre, copper) within heterogeneous networks (e.g., IP, ATM). Media and networks both provide different QoS to the communication subsystem within a computer, QoS-architectures try to cover these differences and offer an abstract interface with certain parameters (e.g., bandwidth, loss-ratio, jitter, delay).

Many QoS-architectures concentrate on how to provide QoS-parameters to applications and how to map different parameters onto each other. Examples are the mapping of application specific parameters, such as frame rates for a video application, onto system specific parameters like CPU time or memory consumption. These parameters are again mapped onto network specific parameters like maximum bandwidth or average delay. Many of these efforts resulted in a large set of rules and functions for mapping parameters depending on system properties, protocols, and applications (e.g., Campbell, 1994).

However, almost none of these advanced systems is in widespread use due to several reasons:

- History tells, that quite often the simple systems survive, although they are not always better from a technical point of view. Examples are Ethernet and IP, both simple "best-effort" technologies, but still fulfilling their main purpose. The add-on offered by technologies like TokenRing or ATM is not always worth the higher complexity. Although ATM as a more advanced technology offers guaranteed QoS, QoS in IP based networks seems to be the way many people go. Additionally, market forces often favour simplicity (e.g., Cisco, 1997).
- Many of the new QoS architectures also require new operating systems, new applications or even new hardware to show their potential. Generally, users do not switch to complete new systems besides for special applications (e.g., high quality TV studios). The biggest problem for new architectures is the lack of standard applications, such as word processing, spreadsheets etc. Users do not want to have separate computer systems.
- A major problem from a research point of view is the controllability of complex systems. Today's computer systems do not consist of a single CPU, a single physical memory and execute only one process at a single point in time. Even the simplest PC comprises many processors each with a separate memory. Examples are network adapter, frame-grabber, video adapter, SCSI adapter etc. Many QoS architectures cover only the main CPU, the scheduler of the operating system and the communication oriented applications. Interrupts by network events are hard to control, copying between bus-master adapters cannot be predicted, non-communication oriented programs, such as, e.g., a compiler, are neglected. A manageable QoS architecture can, thus, not cover all system parameters, but still has to try to supervise the system precisely and to apply the correct mapping functions. Thus, these approaches often try to apply exact rules on fuzzy parameters to get exact results. Quite often unexpected system events render these decisions useless.

This paper proposes a system architecture with the following features that incorporates fuzziness of system parameters, reacts on changes and still tries to give certain guarantees for communication applications:

- The architecture is simple and does not require a large set of system parameters.
- Existing applications still work, the proposed system enhancements can be integrated into COTS (commercial of the shelf) operating systems (e.g., WindowsNT, WindowsCE, Unix) and standard computer architectures (PCs).
- Users must be "relieved" from complicated new features, applications have to adapt autonomously to changes, users can utilise the enhancements via a very simple interface.

The remainder of the paper is organised as follows. The next section discusses QoS mechanisms in more detail and presents a general system architecture based on two internal feedback loops which can be integrated into common systems and incorporates the idea of fuzzy parameters. The third chapter shows more detailed one main component within the architecture, a packet scheduler for QoS provisioning in the lower communication layers. Finally, the conclusion outlines key features and the next steps towards a complete implementation of the proposed architecture.

2 SYSTEM ARCHITECTURE FOR QOS VIA FEEDBACK CONTROLLED SCHEDULERS

Two fundamental approaches dominate solutions for QoS, one is QoS management architectures, the other comprises adaptive applications. While the first typically requires exact knowledge about the operating system, computer architecture etc., the latter assumes the ability of adaptation to changes in resources from applications and protocols.

- *QoS management*: these approaches comprise several mechanisms, such as resource reservation in advance, mapping of QoS parameters, admission control, QoS (re)negotiation etc. If these mechanisms are applied and all resources within systems and networks are considered, this is a way to guarantee QoS from application to application on end-systems. However, several factors make the realisation of these approaches extremely complex and impractical. Due to the worst case reservations needed for hard QoS guarantees, resource usage is typically very inefficient. Furthermore, it is well known that it is in general not possible to predict the resource requirements for multimedia applications. Additionally, many QoS management architectures require changes to user interfaces and applications.
- *Adaptive applications*: this approach, typically, does not give hard guarantees for QoS. Every application starts with an initial amount of resources and tries to provide the best QoS possible based on the actual available resources (Gecsei, 1997). It is quite obvious that this approach cannot offer constant or predictable QoS for users.

2.1 Feedback mechanisms for QoS provisioning

To circumvent the disadvantages of traditional QoS management frameworks, newer approaches suggest simple feedback mechanisms and a set of adaptive resource managers (Diot, 1997) (Lakshman, 1997). The basic idea is to use

feedback mechanisms within the operating system to provide QoS without exact knowledge about the actual system load or the underlying hardware. Every application providing QoS compares its own requirements continuously with the real offered QoS and generates a feedback signal out of the difference. Based on this signal the operating system redistributes the resources assigned to applications.

These approaches do neither require pre-calculated resource requirements for applications nor do they need complex QoS (re)negotiation. This avoids the disadvantages of complex frameworks and does not necessarily require changes to existing applications. Clearly, these approaches cannot give hard guarantees. In overload situations a decrease in QoS will be experienced, which is inevitable in systems without hard resource reservations. Furthermore, these end-system centric approaches cannot control the whole distributed system with feedback loops between different applications on different end-systems.

However, for many users this "soft" QoS on their own end-system is satisfactory. Users will notice at once performance degradation caused by themselves and can then decide how to react, e.g., stop the new application, accept the degradation, or stop another application. Co-existence of hard and soft QoS mechanisms is possible, but out of scope of this paper. Controlling the whole distributed system requires similar mechanisms on all end-systems participating and appropriate control loops in-between. This does not seem to be realistic as a first step in a world of highly heterogeneous systems, networks, and applications.

2.2 Layered feedback loops

A closer look into the system architecture of end-systems shows, that the implementation of a single feedback loop is not useful due to the very different time scales. For example, the scheduler and traffic shaper presented in chapter 3 works within microseconds, CPU-schedulers are typically based on milliseconds while interactions with users happen in the second to minute range. Thus, this work identifies three different layers within a system, separated by the different timing requirements for changes within the layer (c.f. figure 1). The layers are then coupled via feedback loops as explained in the following.

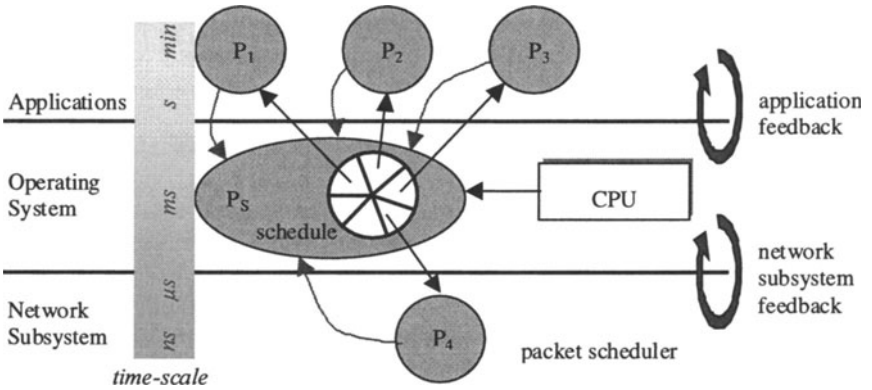

Figure 1 Layered feedback loops.

The three layers are the application layer with application processes, the operating system layer with the scheduler and the network layer with the packet scheduler and integrated traffic shaper. This architecture decouples time-critical events in the network layer from longer term changes in the application layer. Management of the CPU as the central resource is done via the scheduler which is in-between the two feedback loops.

While chapter 3 focuses on the lowest layer and presents an already fully implemented software solution to provide QoS, the following gives some details regarding the feedback loops in general.

Network subsystem feedback

The packet scheduler of the network subsystem determines which packet, or in the case of ATM which cell, will be sent at what point of time. A software based approach as presented in chapter 3 will need a certain number of CPU cycles to perform this task. If the scheduler cannot get the required cycles, packets will be delayed. Based on this delay a feedback signal is generated and returned via a (software) low-pass filter to the CPU scheduler. The CPU scheduler can now assign more CPU time to the packet scheduler, enabling it to perform the scheduling closer to an optimum. This feedback can be extremely fast and without any user intervention. Additional overhead for determining actual system load is not necessary.

Application feedback

In a similar manner a feedback is constructed between applications and the CPU scheduler. Now two inputs are possible. One input comes from the application itself noting that it needs more cycles. This can be done via measuring, e.g., playout buffers. This mechanism is necessary to maintain a certain level of quality without any user intervention. The second input comes from an unsatisfied user. A system offering users too many controls and parameters to change the system behaviour will not be accepted. However, integrating a simple "**Q**uality" button for the user to express his or her dissatisfaction is an easy to understand interface (c.f. figure 2). If the user hits this button, the CPU scheduler tries to assign more CPU cycles to the appropriate application and the processes on which this application depends. Pressing this button while holding the "shift"-key frees CPU cycles from this application, thus, potentially rising the performance of the other applications. This is needed if a user is dissatisfied with the performance of the other applications controlled by the scheme. Using more complex controls, e.g., sliders, requires a clear semantics behind shifting the slider. It is not at all obvious for a user to understand the implications of shifting a slider half-way up or 10% down for a video or audio application.

Figure 2 Simple user interface via a **Q**uality button.

The application shown in figure 2 is up to now a very simple test module which can generate a signal based on the **Q**uality button to increase/decrease the CPU time associated to the process while copying data to/from another end-system. For this first step only standard operating system calls of WindowsNT have been used to influence the scheduler for it is not possible to change the built-in scheduling mechanisms themselves.

It is obvious, that these mechanisms always try to steal cycles from other applications in case of overload situations to favour an application the user determines. Assuming finite resources and no detailed knowledge about the underlying system this is the best one can do without complicated user interaction.

The big advantage of this "soft" QoS solution is its applicability to any system, from a low performance PDA with WindowsCE to a standard PC running UNIX or WindowsNT without changes to standard applications or the operating system. Furthermore, it will be easily accepted also from non-technical users who do not understand the implications of, e.g., changing resolution or colour depth of a video conferencing tool, switching between different audio encodings etc.. Adaptive applications necessary for this approach are already available, examples are vic (Busse, 1996), vat (Jacobson, 1995), IVS (Bolot, 1994) etc.

3 SOFTWARE-BASED PACKET SCHEDULING WITH ADVANCED SHAPING

While up to now a general architecture for QoS provisioning was discussed, the following presents detailed mechanisms for the packet scheduler residing in the lower layer of figure 1. This process is crucial for the provision of QoS and is typically implemented in hardware for ATM interfaces (so called traffic shaper chips) or missing in the case of best effort adapters (e.g., Ethernet).

Today's ATM networks and the future Internet both support QoS. Many discussions are going on how to provide what kind of QoS. However, it is quite clear, that basic support of packet scheduling and traffic shaping is necessary. While many adapters for the Ethernet simply push packets ready to send onto the network and, thereby, create very bursty traffic, enhanced systems have to create a certain shape of traffic to stay within limits of some traffic contract. The following discussion is based on the ATM terminology, because up to now it is not so obvious what parameters will be defined within an enhanced Internet. However, this paper shows, that scheduling and shaping is feasible for ATM for a high bitrate, thus, one can conclude that this will be even easier for the much longer IP packets compared to the short ATM cells.

3.1 Existing hardware shapers

Existing shaping solutions are typically based on dedicated hardware chips. 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 data, but rather on the algorithms, data structures, and available processing power for traversing connection information.

Furthermore, 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).

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.

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. For example, 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, 1997) which uses GCRA (Generic Cell Rate Algorithm) 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. Altogether, this discrepancy may have the effect of cell-loss at the UNI although sender and UPC agreed upon the same traffic parameters in the traffic contract.

3.2 Software process for shaping and scheduling

The CPU has to run a process for packet scheduling and traffic shaping. This process makes sure that the right cell will be sent at the right time according to the traffic contract. The shaper function manipulates context information for all connections, i.e., Peak Cell Rate, Sustainable Cell Rate, Minimum Cell Rate, number of PDUs etc. This function also provides the scheduler function with the necessary timing information for the next cell of the current connection, i.e., the earliest point in time this next cell can be sent without violating the traffic contract. Using this information the scheduler function updates its sorted list of connection identifiers (realized as a heap without pointer operations). The criterion for sorting is the time the next cell of a connection can be sent. The scheduler function returns a pointer to the context data for the connection on top of the sorted list. This selection of the next eligible cell has to be done within, e.g., 2.8µs for a 155Mbit/s

ATM adapter if there were no other processes running. For more technical details see (Schiller, 1998).

The shaper function is based on a combination of the VSA (Virtual Scheduling Algorithm) and LBA (Leaky Bucket Algorithm), both proposed as possible algorithms for the GCRA at the UNI to a public ATM network (ATM Forum, 1996). The implementation presented uses these algorithms for actively creating (and thereby shaping) traffic instead of controlling traffic as UPC at the UNI. If the parameter for delay variation CDVT (Cell Delay Variation Tolerance) is set properly, this approach guarantees the acceptance of cells at the UNI.

3.3 Priorities and Proportional Sharing

A very important question is how a scheduler 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 (variable bit-rate) 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 compared to connections fetching pictures from Web-pages. This would result in a higher audio quality and only minimal additional delay for the picture data transfer.

Fairness in overload situations is an important feature of a scheduler. 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. Thus, the implementation fulfills the criterion of ideal fairness as defined in (Varma, 1997). To privilege one application, one can shift the application to a higher priority. The implementation guarantees this proportional share within one priority class independently for every connection. The settings of the proportional sharing between different priority classes is left to the operator and depends on the policy of a service provider.

Prioritizing an application leads to 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. The

implementation allows both alternatives by guaranteeing proportions of the total bandwidth in an overload situation.

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).

3.4 Best effort services

The scheduler implemented supports not only CBR (constant bit-rate) or VBR in an ATM context. Via the priority and sharing mechanisms it is also possible to support best effort services like UBR (Unspecified Bit Rate) and adaptive services like ABR (Available Bit Rate), (ATM Forum, 1996). UBR traffic is given a PCR rate, which could be below the actual link rate to avoid unnecessary cell loss downstream. If there is UBR traffic to send, it will use idle cell slots, but back-off as soon as other traffic sources have data to send or its PCR limit is reached. UBR traffic can be handled by assigning the lowest priority to this traffic class. The PCR could be dynamically set.

ABR does not need a new mechanism, since it can be seen as VBR with dynamically adjusted rate. In ABR, a minimum user requested rate is guaranteed. As opposed to UBR, ABR should get a fair share even in transient overload situations. Thus, ABR traffic is assigned to a higher priority class than UBR. The ABR feedback mechanism is independent of this work and the PCR is set dynamically in the connection states according to the feedback channel. The shaper and scheduler will then automatically adjust the rates.

A straightforward priority set-up is to give CBR traffic the highest priority, followed by VBR-rt (real-time), VBR-nrt (non real-time), ABR, and finally UBR with the lowest priority. The algorithm can now guarantee the different classes a minimum share, except for UBR which does not need one. Within priority classes proportional sharing is applied, every priority class can get the full bandwidth if no cells from other connections at higher priority levels are ready for transmission.

3.5 Scheduling variable sized packets

The basic scheduling mechanism is also applicable to variable packet sizes. One example of variable packet sizes is the scheduling of IPv6 packets, where the *flow id* is used as a connection identifier. One way to schedule variable sized packets is to change the parameters used in the algorithm as follows. Rates are expressed in bytes/s instead of cells/s, and the earliest sending time is adjusted according to the size of the packet. The tokens in the LBA now count bytes instead of cells, which ensures that a connection gets bandwidth up to its share and also the long term fairness. Variable packet sizes will inevitable introduce delay variance when multiplexed. Once the adapter has started to send a packet it cannot be interrupted and will block other, possibly shorter and more urgent packets. For all networks there exists a maximum packet size, e.g., 1500 bytes for Ethernet, which puts an upper limit on the delay caused by multiplexing and blocking.

If there are enough tokens in the bucket, the packet is eligible for sending as soon as possible. The scheduler should then find the earliest sending time for the connection which depends on already scheduled packets for other connections. If there are not enough tokens, the scheduler has two alternatives. The first is to postpone the scheduling until there are enough tokens. The other alternative is to calculate when there will be enough tokens and then schedule the packet for that time. The first alternative is not attractive since we need to activate the scheduler again when there are enough tokens available (which is indeed predictable). The second alternative also has a drawback, since it may generate bandwidth internal fragmentation. If a packet is scheduled in the future, it may be the case that earlier empty slots can not be used since they are too small to fit a large packet. This is not the case for ATM cells, since they have fixed sized cells.

Our current algorithm works according to the second principle. A slow connection packet will hence be scheduled far away in the future. There are some unsolved fairness issues here that need to be considered. For example, a slow connection with large packets will have little or no delay variance while a high bandwidth connection may have problems to find a big enough slot and will be delayed. The fairness problem, a detailed evaluation, and performance measurements are part of ongoing work.

3.6 Implementation and performance evaluation

The algorithms were implemented on different platforms, such as PentiumPro, Pentium-II, Alpha, UltraSPARC running the appropriate standard operating systems (WindowsNT, Linux, Digital UNIX, Solaris). All measurements show the number of CPU cycles needed for calculating the sending time for the next cell of a connection, resorting the list of active connections, and initiating the actual transfer of the cell. Absolute upper limits are, e.g., 2.8µs for a 155Mbit/s adapter, 680ns for a 622Mbit/s adapter if no other processes are running. Assuming application processes, this time is certainly reduced. All measurement use worst case scenarios, i.e., the scheduler always had to resort the complete list. Furthermore, running the full operating system guarantees realistic results for the execution times compared to stand alone systems often used for performance figures.

The example in figure 3 shows clearly, that with today's processors this task can be done fast enough and no dedicated scheduling and shaping hardware is needed. In the example 100 connections are scheduled, each connection sends 5 PDUs with 5000 bytes each. Clearly visible in this example are 5 peaks for issuing DMA commands to prefetch new data at the start of a new PDU. For more implementation details see (Schiller, 1998). The measurements took place on a 200MHz PentiumPro running Windows NT 4.0. The average of 270 cycles equals 1.35µs. Assuming a 155 Mbit/s adapter, i.e., generating a cell every 2.8µs in worst case, this example leaves ca. 50% of the CPU power for other processes. Further experiments showed that scheduling and shaping of 64000 individual connections requires 620 cycles on average, 1000 connections require 330 cycles on a PentiumPro architecture. The shaping function is in O(1), the scheduler in $O(\log_2 \#\text{connections})$.

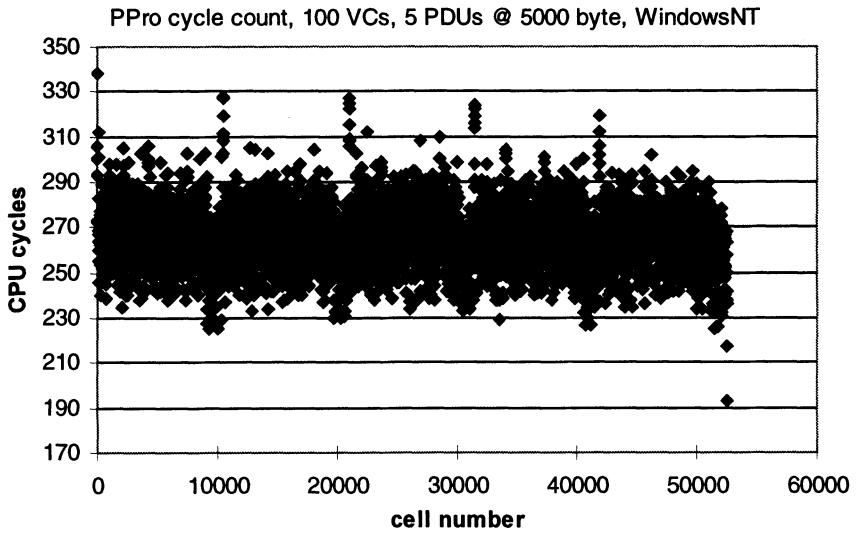

Figure 3 Scheduling and shaping of 100 individual active connections.

4 CONCLUSIONS

The paper discussed basic problems of traditional approaches for QoS provisioning, mainly complexity and incompatibility, and proposed a simple architecture based on feedback controlled schedulers. This approach enables automatic adaptation to changes in system parameters and the environment. The main component for QoS in the network subsystem, a packet-level scheduler with advanced shaping, was explained more detailed. The performance evaluation showed, that it is absolutely feasible with the software approach to reach the throughput necessary for high-performance adapters and to guarantee a broad range of QoS parameters for individual traffic flows in case of ATM. In case of future IP-based networks with differentiated services as currently discussed (Wroclawski, 1998) the CPU load caused by scheduling and shaping is even lower. Thus, one next step of our work is the application of the algorithms to variable sized IP packets and standard Ethernet adapters to provide the shaping necessary for differentiated services.

As discussed in chapter 2 the fully implemented software-based scheduler is only one component within the system. The current very basic add-ons to the operating system scheduler have to be extended and refined. In case of Linux this includes more changes to the scheduler itself, in case of WindowsNT a more powerful hierarchical scheduler on top of the standard scheduler. It is obvious that this approach cannot change commercial operating systems themselves, but has to work on top of them to remain compatible. Furthermore, the up to now basic integration of the **Q**uality-button into adaptive applications has to be improved. Another important topic is the controllability of the feedback loops, especially together with external protocol or application dependant loops (TCP, ABR etc.).

Up to now, decisions are only local for we cannot control the whole distributed system. Here the integration of fuzzy control is under investigation to provide stability and integrate imprecise parameter values. While the whole system may be not as powerful as dedicated architectures with complex QoS management, the big advantage is, that it can be integrated into commercial of the shelf systems.

5 REFERENCES

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

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

Aurrecoechea, C., Campbell, A., Hauw, L. (1998) A Survey of QoS Architectures, Multimedia Systems Journal, Special Issue on QoS Architecture. May 1998

Bolot, J.-C., Turletti, T. (1994) A rate control mechanism for packet video in the Internet. IEEE Infocom, Toronto

Busse, I., Deffner, H., Schulzrinne, H. (1996) Dynamic QoS Control of Multimedia Applications based on RTP. *Computer Communications*. January 1996

Campbell, A. (1994) A Quality of Service Architecture, ACM Computer Communication Review, April 1994

Campbell, A., Aurrecoechea, C.: Hauw, L. (1996) A Review of QoS Architectures, Proceedings of the 4th International IFIP Workshop on QoS (IWQoS 96), Paris

Cisco (1997) Cisco's Packet over SONET/SDH (POS) Technology Support; Mission Accomplished. Cisco Systems, Inc., White Paper

Diot, C., Seneviratne, A. (1997) Quality of Service in Heterogeneous Distributed Systems. Proc. of HICSS'97, 30. Hawai'ian Int. Conf. on System Sciences

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

Gecsei, J. (1997) Adaptation in Distributed Multimedia Systems. *IEEE Multimedia*. April/June 1997

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

Jacobson, V. (1995) Internet Multimedia Systems. Tutorial, ACM SIGCOMM, London

Lakshman, K., Yavatkar, R., Finkel, R. (1997) Integrated CPU and Network-I/O QoS Management in an Endsystem. Proc. of 5th IFIP International Workshop on Quality of Service (IWQoS 97)

Landfeldt, B., Seneviratne, A., Diot, C. (1998) User Services Assistant: An End-to-End Reactive QoS Architecture. Proc. of 6th IFIP International Workshop on Quality of Service (IWQoS 98)

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

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

Schiller, J., Gunningberg, P. (1998) Feasibility of a Software-based ATM cell-level scheduler with advanced shaping. Broadband Communications'98, International Federation of Information Processing IFIP, Chapman&Hall Pub.

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

Varma, A., Stiliadis, D. (1997) Hardware Implementation of Fair Queuing Algorithms for Asynchronous Transfer Mode Networks. *IEEE Communications Magazine*, **35(12)**

Wroclawski, J. (1998) Differential Services for the Internet. http://diffserv.lcs.mit.edu/