Traffic Shaping in End Systems Attached to QoS-supporting Networks

Marc Bechler, Hartmut Ritter:
University of Karlsruhe (TH), Institute of Telematics
Zirkel 2, 76128 Karlsruhe, Germany
mbechler|ritter@telematik.informatik.uni-karlsruhe.de

Günter Schäfer:
Technical University of Berlin, Department of Electrical Engineering
Einsteinufer 24, 10587 Berlin, Germany
schaefer@ee.tu-berlin.de

Jochen Schiller:
Freie Universität Berlin, Institute of Computer Science
Takustraße 9, 14195 Berlin, Germany
schiller@computer.org

Abstract

Quality of Service (QoS) supporting network architectures, like the Differentiated Services architecture, require a certain agreement regarding service levels. Traffic characteristics like data rates will be part of such agreements and, thus, senders must take care to stay within the agreed limits. Traffic shaping is one important mechanism to avoid penalties in networks (dropped or delayed packets) due to violations of the agreement. This paper presents a new traffic shaper based on Class-Based QoS for Linux that aims at shaping aggregate traffic as well as individual flows within an aggregate. In comparison to other approaches (e.g., Class-Based Queuing), our Flow Based Queuing mechanism causes significantly less jitter. Implemented in end-systems, this approach even benefits from direct interaction with applications to create traffic in accordance to application requirements.

1. Introduction

In most QoS supporting network architectures some sort of agreement about the kind of service exists between the network and the end system. In the case of ATM this is called a traffic contract, in the case of the Differentiated Services (DS) architecture similar issues are handled in the service level agreement (SLA) and traffic conditioning agreement (TCA). In general, these agreements define what kind of traffic the network accepts from the end system. A basic parameter in this context is obviously the data rate. If packets are sent from the end system into the network that violate the agreement, the forwarding may be denied or at least not guaranteed depending on the specifications. Thus, an end system has a basic interest in sending well-formed traffic that will not be rejected or penalized at the network border. Therefore, an end system should perform *traffic shaping*. There are additional advantages of traffic shaping in an end system:

- The sharing of the available data rate among different applications can be directly influenced by the end system or the user.
- Considering non-technical issues like pricing, traffic shaping in the end system has additional advantages. In a volume-based pricing scheme, most likely with different prices for different service classes, the user is interested in sending only with data rates and in using only service classes, she or he is willing to pay for. Traffic shaping is the foundation for taking this issue into account, though pricing itself is not further addressed in this paper.

In this paper we present the concept and implementation of a traffic shaper that integrates these requirements. Therefore, the next section gives a more detailed view of the Differentiated Services architecture and the need for traffic shaping. In section 3 two basic design principles of traffic shaping in the end system are discussed, whereas in section 4 a new traffic shaper for class-based QoS-supporting networks based on the existing QoS framework in the Linux operating system is presented. Section 5 describes details of the implementation, in section 6 a competing approach for traffic shaping, based on existing elements, is sketched. The final sections present comparative measurements and a conclusion.

2. Traffic shaping in DS networks

Primary goal of the Differentiated Services (DS) architecture is to provide a simple, efficient, and thus scalable mechanism that allows for better than best effort services in the Internet [1, 2]. In the DS architecture traffic differentiation in the interior network nodes is not based on single applications or flows, as it is the case with ATM

switches during call set-up. In DS a user has a service level agreement (SLA) with, e.g., an ISP (Internet Service Provider), and the ISP gives service guarantees based on aggregates of traffic. The special treatment of different types of traffic is based on the PHB (Per Hop Behavior) of all nodes within a DS domain. Each node in a DS domain only has to check the DS field (which carries the DS code point) and applies simple rules (e.g., remarking the packet, inserting it into special queues) before forwarding. DS boundary nodes classify and possibly condition ingress traffic in addition to the application of the PHB. Traffic conditioning of ingress traffic is based on a TCA (Traffic Conditioning Agreement, part of the SLA) and may comprise metering, shaping, policing, and remarking of the DS field. The DS boundary node can be an ingress router of an ISP or can be immediately co-located with an end system. For example, an ISP may offer 5 Mbps total data rate to the customer and inside the 5 Mbps a Premium Service up to the rate of 2 Mbps. As long as the user stays within 2 Mbps with Premium Service traffic, the ISP guarantees special treatment. Premium Service traffic exceeding 2 Mbps will be dropped.

Now assume the following scenario: A user runs three network applications and decides to favor one of them (e.g., by assigning a higher data rate or a lower delay). However, after the aggregation of all flows the DS domain cannot distinguish different applications or different hosts, although the packet flows have a different importance for a user and should be treated differently. Signaling the "importance" of flows as done in RSVP or ATM (via resource reservation) is problematic due to the well-known scalability and complexity problems in large-scale networks. In [2] it is stated that traffic sources may perform traffic classification and conditioning. Traffic originating from the source domain across a boundary can be marked by the traffic sources immediately before leaving the domain which is called *initial marking* or *pre-marking*. The advantage of initial marking is that the traffic source can more easily take an application's preferences into account when deciding which packets should receive a better forwarding treatment. The host can now set the code point in the DS field according to the local importance of an application. In order to ensure conformance of traffic to the TCA, the end system performs traffic shaping. Traffic generated by an application and exceeding the limit for premium service will not be discarded at the next ingress boundary node, as the (shaped) traffic already conforms to the TCA. Thus, the end system performs the functionality of a DS boundary node by marking and shaping traffic from applications running on that host.

3. Design principles of the traffic shaper

The basic task of a traffic shaper within the end system is to share the available data rate among the applications sending data according to their different requirements. For the design of such a traffic shaper, two basic approaches are conceivable:

- All applications specify their required data rate for each flow (and potentially further QoS parameters). On the basis of those requirements the traffic shaper schedules the available data rate. If an application desires a guaranteed quality of service support for its flow, the traffic shaper has to perform a restrictive admission control by observing if the new requirements are reconcilable to all existing guarantees. If there are not enough resources available the new flow will not be admitted. Note that the data rate that was guaranteed to an accepted flow cannot be used by new flows, even if it is unused for a longer time.
- The alternative approach assumes that hard guarantees are not needed in many cases. Additionally, many applications are not able to specify their exact quality of service requirements; applications are more likely to define a minimum data rate that is needed for an acceptable performance. Only if the sum of the minimum data rates of all existing flows is above the available aggregated rate, a new flow does not get any data rate. Apart from the guarantee of a minimum data rate the available data rate will be shared equally between all flows in the beginning. Elements of the higher communication layers are able to act on the sharing in favor of one (or more) flows, resulting in a weighted proportional sharing.

The first design approach achieves hard guarantees by reserving more resources than permanently needed. This approach is especially useful for applications with static requirements, such as controlling machines, where a short disruption of the available data rate may result in the loss of the machine's control or even in damages. The second design approach using a weighted proportional sharing is interesting for multimedia applications. Usually, multimedia applications are by far more tolerant to variations in the available data rate. It is also rather impossible to define a fixed and narrow interval for long-lasting quality of service requirements of multimedia applications. The use of weighted proportional sharing also allows a more dynamic distribution of unused resources to other applications.

Due to this reasons, we favored the second design approach. The proportional allocation of resources is also more intuitive for the user. Up to a specific point of time, the starting of applications results in an impaired performance of other running applications, which is rather accepted by a user than terminating other flows in order to free the resources needed for new flows. In order to realize a weighted sharing, it is very important that the user has the possibility to actively affect the sharing of the resources. We already presented a simple interface to the user in [3].

4. Class-based traffic shaping

The Linux operating system already contains a framework for traffic shaping. On top of the network interface there is usually a simple FIFO queue. All outgoing packets are written into this queue from the higher protocol layers in a first-come-first-served manner. The Linux QoS framework allows to deploy different queuing disciplines besides the basic FIFO queuing discipline, and beyond that it supports the notion of classes and filters. These elements can be loaded or unloaded dynamically from the command line using the Traffic Control (TC) tools described in [4].

Using the Linux QoS framework and the TC tools, a typical tree structure can be built. Figure 1 shows the setup of our traffic shaper supporting DS in the end system. It is based on our newly developed *Flow-Based Classifier* (*FBC*), which assigns data to be transmitted to the service classes Premium Service (PS), Assured Service (AS) and Best Effort (BE). The assignment is based on the premarking of the packets, which is triggered by the applications selecting an initial service class. In IP networks using Integrated Services (and also in ATM networks) the selection of the service class has to be explicitly signaled by the upper layer.

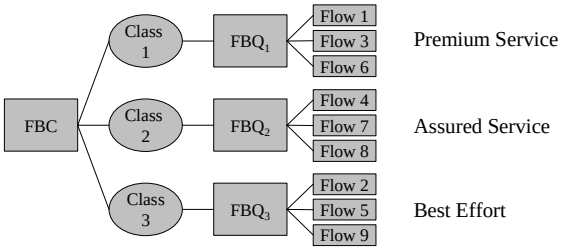

Figure 1. Hierarchical Shaping Approach

Additionally, we implemented a new queuing discipline called *FBQ* (*Flow Based Queuing*). Each service class is associated with one FBQ element. Each FBQ in turn manages one FIFO queue per flow. If a new flow is initialized, the FBQ establishes a new queue for this flow; if the flow is already known, the data that has to be transmitted is enqueued in the FIFO queue related to this flow.

Initially, the traffic shaper shares the available data rate among all existing classes or FBQs in a static way. The sharing depends on the SLA between the end system and the boundary node. After a packet which is passed down at the socket interface is assigned to a flow, the traffic shaper controls if any packets of the flows are ready for transmission. The control starts at the FBC element and is performed recursively for each class, FBQ, and flow. The control sequence of the classes depends on their priority that is defined during the initialization procedure. The FBQ of each class marks packets as ready-to-transmit only if the transmission would not exceed the determined data rate and burst size for this flow. If a packet is ready for transmission, it is dequeued and transmitted; otherwise the traffic shaper calculates the earliest transmission time and starts a timer if there is no other timer running with an earlier deadline. When the timer expires, an interrupt is triggered which recursively searches for packets that can be transmitted.

4.1. Weighted Sharing

The traffic shaper performs the sharing of the resource data rate on two levels. Within an FBQ, the shares of the dedicated flows are continuously adjusted to the overall data rate of this FBQ. At the second level, the unused resources of an FBQ can be used by other FBQs. As described before, this is useful for end systems in DS networks, as there is a high correlation between Assured Service and Best Effort that renders shifting of (parts of) flows between the two service classes useful.

Single flow	f resp. f_i
Set of all flows	F
Single FBQ	q
Set of all defined FBQs	FBQ
Share of the available rate for a flow f_i in FBQ q	$s(f, q)$
Data rate of a FBQ q / a flow f	$d(q) / d(f,q) = d(f)$
Token of a FBQ q / a flow f	$T(q) / T(f,q) = T(f)$

Table 1. Basic definitions used in this paper

With the definitions of Table 1, the share s in the overall data rate of the corresponding FBQ q and the permitted data rate d of a single flow can be described as follows:

(4.1) $\forall q \in F(q),\ F(q) \neq 0: \quad \sum_{f_i \in F(q)} s(f_i, q) = 1$

(4.2) $d(f, q) = d(q) \cdot s(f)$

$\forall q \in F(q),\ F(q) \neq 0: \quad \sum_{f_i \in F(q)} d(f_i) = d(q)$

In the common case, each flow gets the same share on the data rate of $(1/n)$, where $n=|F(q)|$ represents the number of flows in an FBQ. The data packets of flow f are dequeued with the maximum data rate $d(f)$, which results from the share s in the data rate of the FBQ. If an existing flow is terminated or a new flow is added to the FBQ, the share s of the other flows changes accordingly. Thus, a new flow f_{n+1} obtains a share of $1/(n+1)$ on the data rate $d(q)$. The share of all other flows in this FBQ is reduced according to equation (4.3):

(4.3) New flow $f_{n+1} \Rightarrow$

$$f \neq f_{n+1}: s^{t+1}(f) = s^t(f) - (s^t(f) \cdot \frac{1}{n+1});$$

$$f = f_{n+1}: s^{t+1}(f) = \frac{1}{n+1}$$

With the termination of flow f_i the data rate $d(f_i)$ is released and can be used by other flows again.

In correspondence to the design requirements, a minimum data rate can be defined for each flow. If a flow obtains the minimum data rate, traffic shaping must not reduce its share on the data rate any further. If a new flow is added to the FBQ in this case, the number n of flows reduces by the amount of the flows that must not be reduced anymore.

When flows with a minimum data rate are terminated, the released data rate is shared among the remaining flows. With the use of a minimum data rate, the scheduled data rate for a new flow decreases. If the overall available data rate of each flow reaches the defined minimum data rate, i.e.,

$$\sum_{f \in F(q)} d_{\min}(f) = d(q), \quad (4.4)$$

a new flow cannot be accepted without violating the minimum data rate of one or more existing flows. However, the new flow is not rejected; it simply gets a data rate of $d(f) = 0$. By releasing data rates of other flows, the traffic shaper can immediately assign a data rate of $d(f) > 0$ to the new flow, without any interaction with the higher level.

4.2. Interaction with the higher layers

Interaction with the higher layers comprises two features: First, the proposed traffic shaper gives feedback on its internal status to elements in the upper transport layer. Second, the weighted sharing supports acting on the adaptation mechanisms. We first discuss the affect on weighted sharing of the data rate between the existing flows. The basic idea behind this feature is that the transport layer protocol, an application, or a management entity is able to signal the traffic shaper that a flow should be preferred and get a higher share on the data rate (if enough resources are available). A preference of flow f_i can be achieved by increasing its share s on the data rate of the related FBQ. If the upper layer signals that a flow f should be preferred, the percentage for increasing the share of f on the FBQ's data rate is specified by the configurable parameter *incr*.

However, the data rate for one flow can only be increased, if the minimum data rates of all other flows are observed. Thus, the maximum available (free) data rate s_{free} for an FBQ is computed in the following way:

$$s_{free}(q, f_i) = 1 - s(f_i) - \sum_{f_j \in F(q), f_j \neq f_i} s_{\min}(f_j) \quad (4.5)$$

Increasing the share of one flow is done at the expense of reducing the shares of other flows. The limit of this procedure is reached when there is no free share according to equation (4.5). In this case, mechanisms for displacing flows to other classes can be applied. If the share of a flow has been reduced, it is checked if the new share s^{t+1} is below a threshold $thres > d_{min}(f)$, which can be defined for each flow. If $s^{t+1} < thres$, the traffic shaper tries to move the flow to another FBQ.

The second feature of the vertical adaptation realizes the feedback from the traffic shaper to the basic elements that lie above. This feedback signal contains the current status of the traffic shaper and will be sent if the utilization of a flow's transmission queue predefined thresholds. Currently we are examining the use of this signal for an improved adaptation of the TCP protocol.

5. Implementation

We have implemented both FBC and FBQ, using the Linux operating system (kernel 2.3.49). The complete source code is available at [5]. In this paper, we focus on three essential topics of the implementation.

Each outgoing packet is associated with a flow. The start of an application sets up a new flow context. Based on the memory address of the socket structure, this context information is stored in a hash table for all flows. All further packets of this flow are assigned to a context by a lookup in the hash table. Thus, a fast access to all context information of a outgoing packet (priority, initial service class, maximum rate etc.) is realized.

The core of the traffic shaping is based on the well-known token bucket model, which is also used in a related approach described in the next section. Whereas in that approach a token represents a clock tick, a token of the bucket used in FBQ and FBC represent one byte. A common token bucket is determined by the two parameters *fill rate* and *bucket size*.

In the DS scenario, Premium Service (PS) and Assured Service (AS) are considered. Whereas in AS small bursts exceeding the assured data rate are allowed, in PS it is assumed that the maximum rate of a service class is fixed and that any bursts with a rate below the maximum rate are allowed. In order to shape the aggregated traffic of a class in the FBQ, a token bucket with a bucket size of the allowed burst size and a fill rate of the negotiated rate of the service class is used (thus ignoring the slightly enlarged burst size due to tokens arriving during sending). Additionally, each flow inside the FBQ has to be shaped according to its share on the aggregated rate of the FBQ. The burst of each flow is also limited. Thus, traffic shaping is done in two steps: First, the tokens that were added since the last calculation of the tokens for the FBQ q are calculated as follows (t = timestamp of the last calculation of the token bucket, η = timespan since t):

$$\Delta T^{t+\eta}(q) = \Delta t^{t+\eta} \cdot d^{t+\eta}(q) \quad (5.1)$$

Afterwards, the tokens are divided between all flows of the FBQ, according between the weighted share of the flow. The amount of new tokens available for a single flow f_i inside the FBQ is the minimum of this share and

the maximum amount of tokens that could have summed up due to the size $TBsize(f_i,q)$ of the token bucket of flow f_i in this FBQ q ($T^t(f_i,q)$ is the amount of tokens of flow f_i at time t):

(5.2) $$\Delta T^{t+\eta}(f_i,q) = \min((s^{t+\eta}(f_i,q)\cdot\Delta T^{t+\eta}(f_i,q)),(TBsize(f_i,q)-T^t(f_i,q)))$$

A packet of a flow can be transmitted if the flow has enough tokens (*packetsize* denotes the size of a packet in bytes), i.e.,

(5.3) $$packetsize \le T^t(f_i,q)+\Delta T^{t+\eta}$$

Nevertheless, if all n flows exploited their token bucket at once and sent a burst of size *TBsize*, this would result in a burst of the aggregated traffic of size $n\cdot TBsize$. In the case of AS, this should not be a problem as some bursts will most likely be accepted. For PS however, this could violate the TCA depending on its exact specifications, which might even allow some bursts. In order to avoid packets to be dropped, another token bucket limiting the burst size of the aggregated flows of the FBQ is needed. If a packet is ready for transmission (cf. equation (5.3)), it has to be delayed until enough tokens are available in the token bucket of the FBQ. This amount is calculated according to equation (5.4) ($T^t(q)$ is the amount of tokens of the FBQ q at time t):

(5.4) $$tokens = min((T^t(q)+\Delta T^{t+\eta}(q)),TBsize(q))$$

During this delay period no other packets can be sent from this FBQ. Otherwise, it could happen that a large packet is infinitely blocked due to a flow with small packets, which is always preferred.

6. Comparison with Class-Based Queuing

In the context of the Linux QoS framework, many other queuing disciplines have been developed (for an overview see [6]). The most promising approach to realize class-based traffic shaping in the end system using existing queuing disciplines seems to be the class-based queuing (CBQ) [7]. Using class-based queuing, flows can be assigned to classes. CBQ uses a token bucket for each class and polls all classes for data ready to be sent using a round-robin algorithm. As will be shown in the next section, this causes a high jitter, because the interleaving of packets of different classes is very coarse. In Figure 2 a setup of a traffic shaper using CBQ for DS networks is shown.

Adding new flows or terminating existing ones is performed quite frequently in end systems. The structure of CBQ does not support this efficiently. Without modifications, there is just one fifo queue per class; per-flow shaping is not inherently supported. Thus, CBQ might be the right choice for traffic shaping in routers, where traffic aggregates are more static and per flow shaping is not needed, but not in the targeted multimedia end systems. Other approaches like [8] also focus on traffic shaping of aggregates in DS-routers, not on per-flow shaping in end systems.

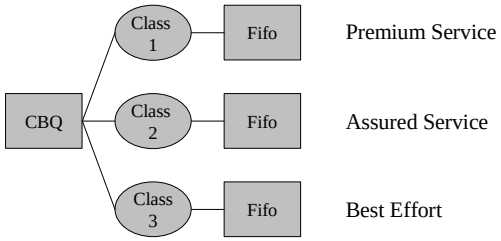

Figure 2. Traffic Shaping using CBQ

7. Results

In order to evaluate our implementation, we analyzed two parameters: The jitter that results from the interleaving of different flows, and the resulting CPU load. We compared our approach using the FBC/FBQ elements with an unmodified kernel realizing one FIFO queue (called RAW approach), and the CBQ-approach described in section 6. The data sources were test applications sending UDP packets with packet sizes of 1400 bytes, running Linux version 2.3.49 on a 450 MHz PC, 64 MB RAM, attached to a 100 Mbps Ethernet. The data rate was divided between the service classes as follows: 10 Mbps Premium Service, 30 Mbps Assured Service and 40 Mbps Best Effort. The traffic shaper was allowed to share unused data rate across classes, but must not exploit the remaining 20 Mbps in order to prevent network congestion.

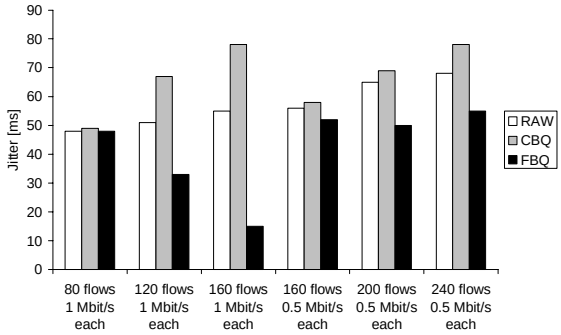

Figure 3. Comparison of the jitter per flow

As can be seen in Figure 3 that shows the jitter of a single flow resulting from the traffic shaping, in all cases the jitter is smaller compared to the CBQ and RAW approach. As said before, the traffic generators were simple applications sending data at UDP sockets. The data packets are not passed to the socket interface in equal time distances due to the multitasking capabilities of the operating sys-

tem, so there is already some jitter when the packets arrive at the traffic shaper. The token bucket size of each flow was sufficient to let the jitter pass, otherwise it would disappear. The traffic shaping did not reduce the data rate, because the sum of the flow rates does not exceed the allocated rate of 80 Mbps. All packets passed to the traffic shaper were immediately sent out.

If the sum of the data rates of all flows is above the allocated rate, CBQ generates a high jitter, due to its simple round robin scheme. In CBQ, as long as packets are in a class and the rate of the class is not exceeded, the packets are sent, thus resulting in a coarse interleaving. The measurements show that FBQ/FBC reduces the jitter significantly. This observation holds especially in the case of 1 Mbps flows. The interleaving of packets of different flows in different classes is fine granular.

In the case of 0.5 Mbps flows, the jitter is not reduced to the same extent. Nevertheless, in all cases the jitter is smaller than using the RAW and the CBQ approach. There seems to be room for further optimizations of the performance in the case of many flows. The observations also result from a coarse multitasking of the great amount of traffic generators. Obviously, the traffic shaper can not send packets before they were passed to it.

As expected, the CPU load resulting from the complex algorithms of the FBC/FBQ approach increases compared with the CBQ approach. The measurements were done by taking the time the CPU runs in kernel mode.

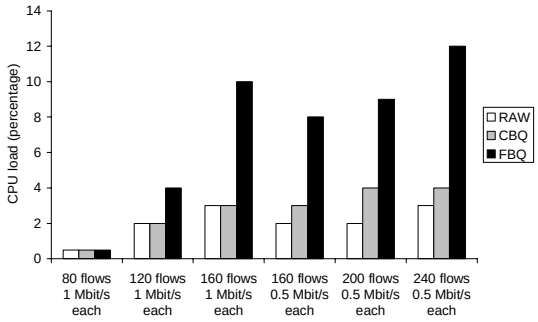

Figure 4. Comparison of the CPU load

Nevertheless, it can be seen in Figure 4, that in the worst case the CPU consumption due to FBC/FBQ was four times higher than the consumption without any modifications (RAW), and only three times higher than the consumption of the CBQ approach. However, an overall CPU load of about 12 percent should be acceptable, especially as this percentage will be reduced with a higher CPU speed. We consider a smaller jitter far more important than CPU load as CPU performance permanently increases rapidly while jitter will always remain problematic.

8. Conclusion

In this paper we presented a traffic shaper for end systems attached to networks with class-based QoS support. It aims at both shaping the aggregated traffic and shaping of each flow inside an aggregate. The basic design of the proposed traffic shaper enables the support of short-term dynamic changes of QoS requirements, as especially applications in end systems are started and terminated quite frequently. The FBC/FBQ framework supports both the weighted sharing of resources inside an aggregate as well as between service classes, if the structure of the service classes allows this. The jitter could be reduced, though there is obviously a tradeoff between fine-granular traffic shaping and increased CPU load.

Future work has to be carried out in detailed measuring of the performance in different load scenarios. Conceptual work has to be done in the area of determining an initial configuration of the traffic shaper and the tuning of configurable parameters. Depending on the relation between service classes, existing service level agreements, and last but not least the applied pricing mechanisms, some decisions like borrowing data rate between classes may be deployed or not. The development in this area strongly depends on the future development of QoS architectures like Differentiated Services, which are not yet stable.

References

[1] K. Nichols, S. Blake, F. Barker, D. Black: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, IETF, 1998

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss: An Architecture for Differentiated Services. RFC 2475, IETF, 1998

[3] M. Bechler, H. Ritter, J. Schiller: Integration of a Traffic Conditioner for Differentiated Services in End-systems via Feedback-loops. Broadband Communications '99 (BC99), Hong Kong, November 1999

[4] W. Almesberger: Linux Network Traffic Control - Implementation Overview. April 1999, available at: http://icawww1.epfl.ch/linux-diffserv/

[5] Source code for the FBC/FBQ available at: http://www.telematik.informatik.uni-karlsruhe.de/~ritter/forschung/ADES/Traffic-Shaping.html

[6] S. Radhakrishnan: Advanced Networking Overview Version 1. University of Kansas, September 1999, available at: http://qos.ittc.ukans.edu/howto/index.html

[7] S. Floyd, V. Jacobson: Link-sharing and Resource Management Models for Packet Networks. IEEE/ACM Transactions on Networking, Vol. 3 No. 4, August 1995

[8] W. Almesberger, J. H. Salim, A. Kuznetsow: Differentiated Services on Linux (draft-almesberger-wajhak-diffserv-linux-01.txt). June 1999, available at: ftp://lrcftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt