IP Traffic Control on UMTS Terminal Equipment

Manuel Ricardo, Rui Soares, Jaime Dias and José Ruela
FEUP – Fac. Eng. Univ. Porto. Rua Dr. Roberto Frias, 4200-465 Porto, Portugal

INESC Porto. Praça da República 93 r/c, 4050-497 Porto, Portugal

Abstract: The paper presents the architecture of an UMTS terminal equipment optimised for IP based communications and describes the traffic control mechanisms required for supporting emerging 3G services.

Key words: IP, UMTS, Quality of Service, Traffic Control, Scheduling, PDP Context

1. INTRODUCTION

This paper presents some results of the work carried out in the European IST project ARROWS (Advanced Radio Resource Management for Wireless Services) [1]. This project aims at providing advanced Radio Resource Management (RRM) and Quality of Service (QoS) management solutions for the support of integrated services within the context of Universal Terrestrial Radio Access (UTRA). The project addresses packet access, asymmetrical traffic and multimedia services, all based on IP. The main objectives of ARROWS are: 1) to define and simulate RRM algorithms for an efficient use of the radio resources; 2) to provide QoS bearer services for packet switched flows at the UTRA; 3) to demonstrate the benefits of the proposed algorithms and procedures by means of an IP based multimedia testbed [2].

This paper is related to the third objective. The ARROWS multimedia testbed consists of the following functional blocks: 1) an all-IP based UMTS terminal; 2) an UTRAN (Universal Terrestrial Radio Access Network) emulator, implementing the UMTS (Universal Mobile Telecommunications System) radio interface and the relevant RNC (Radio Network Control) functions; 3) a gateway implementing functions traditionally assigned to

SSGN (Serving GPRS Support Node) and GGSN (Gateway GPRS Support Node); 4) a backbone IP network and its associated routers; 5) a server.

Only the first functional block is addressed – the all-IP UMTS terminal, which supports multimedia applications and is implemented in a LINUX based PC. Its architecture, the selected applications, the classification, scheduling and shaping of IP flows as well as the interface with the UMTS network interface, assumed to implement the UMTS Non-Access Stratum (NAS) functions, are described.

The paper is organised in seven parts. Section 2 introduces the selected multimedia applications that will run on the terminal. Section 3 discusses the UMTS terminal architecture, which is biased towards the compatibility between the IP and UMTS worlds from flow and QoS points of view. Section 4 reviews the IP QoS facilities currently available in Linux and presents a strategy for using them in an all-IP UMTS terminal. Section 5 describes the mechanism proposed for controlling the traffic. Section 6 gives experimental results that validate this strategy. Finally, Section 7 presents the main conclusions.

2. MULTIMEDIA APPLICATIONS

Four traffic classes were identified in UMTS: conversational, streaming, interactive and background [3]. One main aspect that distinguishes these classes is how delay sensitive the traffic is. The conversational class is meant for very delay-sensitive traffic, while the background class is delay tolerant. Moreover, when comparing the conversational and streaming classes, the former requires a tight bound on delay and stringent control of the delay jitter. This is mostly due to the fact that conversational traffic is symmetric, while streaming is highly asymmetric and therefore it is possible to use buffers for smoothing out jitter. In conversational services, this would increase the delay acceptable for a natural human conversation, turning the communication awkward.

One application representative of each UMTS traffic class was chosen for the ARROWS testbed: Videoconference (conversational), Video streaming (streaming), Web browsing (interactive) and Email (background). All applications were required to satisfy three characteristics: 1) be widely used; 2) be open source, so that extensions to IPv6 or that incorporate new QoS features, for instance, could be easy; 3) have port for LINUX.

For Videoconference, VIC and RAT, the well-known video and audio conferencing tools, were selected as departing applications. Although designed for multicast environments, they are configured in ARROWS as

point-to-point (unicast). Both applications rely on the Real Time Transport Protocol (RTP).

VIC supports the H.261 and H.263 video codecs. RAT supports various codecs, such as G.711 PCM (64 kbit/s), G.726 ADPCM (16-40 kbit/s), LPC (5.6 kbit/s) and GSM (13.2 kbit/s). When used for an audio-video telephony call, these applications generate two real-time and bi-directional IP flows (audio and video) that have to be adequately transported through the UMTS transport services, that is, Radio Access Bearers (RAB) and Packet Data Protocol (PDP) contexts, as discussed in Section 3.

For Video streaming it was decided to use a coding scheme that generates two streams with base and enhancement information, respectively. The application that will be used is the one developed by the MPEG4IP group, which also deploys video over the RTP/UDP/IP protocol stack. When playing a stream, two unidirectional and real-time IP flows have to be transported over the UMTS network.

Web browsing at the terminal requires a browser, that is, an HTTP client. Email, at the mobile terminal, requires both POP3 and IMAP clients. The three application protocols (HTTP, POP3 and IMAP) use the TCP/IP stack. No real-time requirements are envisaged for the IP flows they generate.

Mozilla is being used as departing point for these applications.

3. TERMINAL ARCHITECTURE

The proposed UMTS terminal architecture that supports IP based services is shown in “Figure 1”.

3.1 Functional blocks

Forward includes the IP look-up routing tables and the encapsulation of transport level segments in IP datagrams.

Classifier filters the packets and places them in different queues according to their characteristics. TCP/UDP and IP header fields, such as source port or source IP address, can be used as criteria to classify the packets.

Shaper, simply said, consists of a queue for an IP flow. A queue has properties associated with it, such as bandwidth. A flow must be shaped so that it does not violate the QoS previously negotiated for the associated PDP Context (in the *NAS Module*).

Mapper is responsible for negotiating the activation, modification and deactivation of PDP contexts, passing to the *NAS Module* the desired QoS

parameters. The mapping between RSVP QoS parameters and PDP context QoS parameters is performed by this block.

RSVP implements RSVP (Reservation Protocol) [4] that, in ARROWS, is used to guarantee end-to-end QoS to IP flows that traverse both the UMTS and the IP backbone networks [5]. It can, in some circumstances, be avoided.

NAS module implements Non-Access Stratum functions, such as session management and mobility management. It consists of two planes. On the user plane, the module is offered as a standard LINUX network interface (*umt0*, in "Figure 1") and is able to exchange datagrams with the IP layer. On the control plane, the NAS module is offered as a character device driver (*/dev/nas0*, in "Figure 1"), through which messages for establishing and terminating RABs are exchanged.

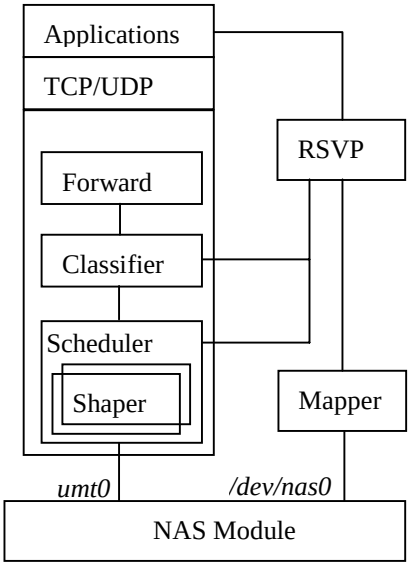

Figure 1 – UMTS Terminal Architecture

3.2 PDP context

A GPRS (Generic Packet Radio Service) subscription consists of one (or more) PDP addresses that, in the case of "Figure 1", will be the IP address associated to the *umt0* interface. Each PDP address, in GPRS, is described by one or more PDP contexts [6]. Each PDP context is associated to a RAB. When more than one PDP context exists, the other PDP contexts must have a TFT (Traffic Flow Template) associated. A TFT consists of up to eight packet filters. Each filter contains a valid combination of the following

attributes: Source Address and Subnet Mask, Protocol Number (IPv4) / NextHeader (IPv6), Destination Port Range, Source Port Range, IPSec Security Parameter Index (SPI), Type of Service (IPv4) / Traffic Class (IPv6) and Mask, Flow Label (IPv6). A PDP context and a RAB, due to their one to one relationship, are used interchangeably in the paper.

The mobile station should be able to support more than one PDP context simultaneously and to forward IP packets into the appropriate RAB. This justifies the use of the *Classifier* and the *Shaper* in "Figure 1". Therefore, more than one PDP context may exist, each with different QoS parameters, to which packets will be forwarded depending on the class of the traffic they belong to. In videoconference, for instance, image and voice are carried as two IP flows. These flows have different QoS requirements and thus may be mapped to separate PDP contexts; synchronisation between the flows is handled at the application level.

RSVP messages are themselves carried on IP datagrams. A RAB for best effort traffic can be used to transport these messages. This RAB can also be used to transport the IP flows for the Web browsing and Email applications.

4. IP TRAFFIC CONTROL IN LINUX

LINUX kernels have been growing to include a number of advanced networking features such as firewalls, QoS and tunnelling. The QoS support, available since kernel 2.1.90, provides a number of features. The working principle adopted in recent kernels (e.g., 2.4.4) is shown in "Figure 2".

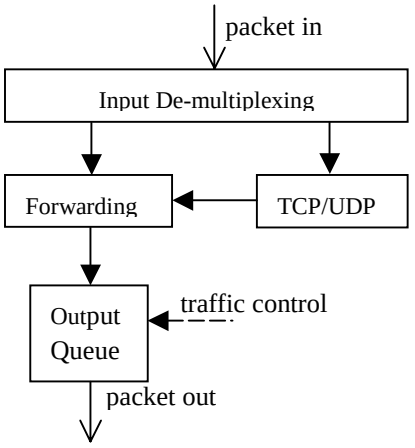

Figure 2 – LINUX traffic control

The *Input De-multiplexing* module examines an incoming packet and determines if it is addressed to the local node. If so, it is sent to higher layers (TCP/UDP block) for processing. Otherwise, it is passed to the *Forwarding* block. This block looks up the routing table and determines the next hop for the packet. The packet is then placed in a queue kept for the device (e.g., *eth0* or *umt0*). Traffic control is performed on this *Output Queue*, just before the packet is sent to the network interface. The traffic control in LINUX consists of three building blocks: *Queuing Discipline*, *Class* and *Filter*.

A *Queuing Discipline* is a framework used to describe a policy for scheduling output packets. It is usually associated to a network interface such as *eth0* or, in the case of "Figure 1", *umt0*. LINUX provides several disciplines for scheduling packets such as FIFO (First In First Out), TBF (Token Bucket Flow), CBQ (Class Based Queuing), RED (Random Early Detection) and TEQL (True Link Equalizer). A *Queuing Discipline* may have a complex structure; a set of *Classes* and *Filters* may be associated to a root *Queuing Discipline* where *Filters* are used to assign packets to *Classes*.

In this case, a *Class* must have associated a new *Queuing Discipline* that, in turn, may have more *Classes* and *Filters*. This principle allows the combination of *Queuing Disciplines*, *Classes* and *Filters* in an arbitrary hierarchical structure. Each network interface may own one of these complex structures.

Filters are then installed on *Queuing Disciplines* to direct packets to *Classes* on that *Queuing Discipline*. The following filters may be used in LINUX: *u32*, *rsvp*, *fw*, *route* and *tcindex*. The *u32* generic filter allows classification of packets based on header fields, such as IPv4, IPv6, TCP or UDP. The *rsvp* filter allows the classification of packets based on the parameters that define an RSVP flow such as the IP destination address and either the port or the Flow Label.

The traffic control may be configured in user space using the *tc* command available in the *iproute2* package. A good description of this command as well as on scheduling, queuing disciplines, filters and traffic control in general may be found in [7], [8].

5. TRAFFIC CONTROL ON THE UMTS TERMINAL

After a PDP context has been negotiated and the associated RAB established, the terminal may start communicating. However, the terminal may have more than one PDP context activated and more than one RAB established, each with its own QoS parameters. It is, therefore, necessary to

direct the packets to the proper RAB, schedule the packets according to their priorities and shape the traffic so that the flow sent to a RAB is compliant with the QoS previously negotiated for that RAB.

5.1 Scheduling

Scheduling is usually required when there is the need to share a link with limited bandwidth. Flows with higher priorities must be scheduled first, taking care that flows with lower priorities do not starve. Thus, the concepts of sharing and priority hold when dealing with scheduling. A flow with high priority may require less bandwidth than another flow with lower priority. Sharing is about bandwidth, priority about delay and jitter.

The services introduced in Section 2 can be ordered in ascending priority as Email, Web Browsing, Video streaming and Videoconference. However, a Video streaming session may require a larger bandwidth than a pure voice session.

The CBQ (Class Based Queuing) discipline [9] can be used to solve the priority issue. This discipline is based on statistical scheduling and a hierarchical tree of traffic classes. When a packet is received, it is classified and associated to a leaf class. It is possible to associate bandwidth and a priority to each class. The CBQ queuing discipline is delivered with two schedulers: generic and link sharing. The generic scheduler must guarantee a low delay to real time flows. The link sharing scheduler tries to avoid that real time flows monopolize the use of the link.

5.2 Shaping

The rate of a flow can be regulated using shaping techniques. In this case, the traffic passed to a RAB needs to conform to the bandwidth previously negotiated for that RAB. The use of a CBQ class for this purpose is not adequate, since none of its schedulers addresses this problem. Better results can be achieved if a TBF (Token Bucket Flow) queuing discipline [10] is associated to each leaf class.

The TBF consists of a buffer (bucket), filled with virtual pieces of information (tokens) at a specific constant rate (token rate). An important parameter of the bucket is its size, that is, the number of tokens it can store. Each token in the bucket lets one incoming data octet to be sent out of the queue and is then deleted from the bucket.

Associating this algorithm with the two flows (token and data) gives three possible scenarios: 1) the data arrives into TBF at a rate equal to the

token rate, which means that each incoming packet has a matching token and passes the queue without delay. 2) the data arrives into TBF at a rate lower than the token rate and therefore only some tokens are deleted when data packets are sent out. Tokens accumulate up to the bucket size and can be used to send data above the token rate, if this situation arises. 3) the data arrives into TBF at a rate higher than the token rate. In this case, incoming data can be sent out immediately, while the token bucket is not empty. The accumulation of tokens allows short bursts of data to be passed without delay and loss, but any lasting overload will cause packets to be constantly dropped (or delayed).

In any case, the average data rate is bounded by the token rate.

5.3 Proposed configuration

A generic traffic control configuration proposed for an IP based UMTS terminal is shown in "Figure 3". Scheduling and shaping of the flows are implemented with CBQ and TBF queuing disciplines, respectively. The location of the filters is also shown. For each RAB, one leaf class on the CBQ queuing discipline (e.g., 1:11) is created. Each class has one TBF queuing discipline associated instead of the generic one installed by default.

For the sake of generality and to prove the flexibility of the solution, four leaf classes are drawn. To support the services presented in Section 2, a finer assignment is required. Videoconference requires two classes, corresponding to two RABs and serving two flows, one for audio and the other for video. Video streaming also requires two classes, for the base and enhancement flows. The other flows, corresponding to Web Browsing, Email and generic signalling, such as RSVP, may be assigned to a fifth class which, in turn, can be mapped to the primary PDP context. No values are presented for configuring the buckets (token rate and bucket size) since they will depend mainly on the cost of the radio channels. However, the proposed solution is flexible enough to fully support dynamic configuration of these values.

Once the packet is sent to the network interface (NAS module, *umt0*) how can it know to which RAB the packet belongs to? The solution proposed is to use the Flow Label, in IPv6 or the Type of Service (ToS) in IPv4 to distinguish the RABs. The TFT associated to a RAB can be given a list of ToS or Flow Label values that belong to each RAB. In this case a given ToS or Flow Label would always belong to the same RAB.

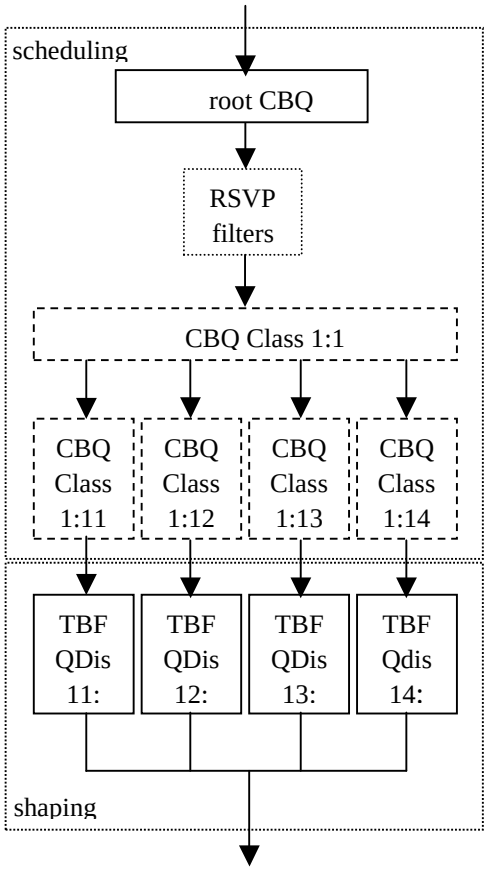

Figure 3 – UMTS terminal traffic control

5.4 Configuration Example

The traffic control may be configured either using *netlink* sockets, if it is an application configuring it, or with the use of *tc* command from *iproute2* package. An example of a script used to configure a hierarchy similar to "Figure 3", but for device *eth1*, is shown below.

Only two classes are configured, with token rates 64 and 32 kbit/s, respectively. The classes have the same priority and are configured for very small IP datagrams – 72 octets, on average. This parameter is required to calculate the variables of the CBQ algorithm. The bucket size is 1500 octets. Finally, two *u32* filters are installed in the root CBQ queuing discipline. Classification is based on the ToS value – packets with ToS values of 0x20 and 0x00 are classified into the 64 and the 32 kbit/s classes, respectively.

```
#!/bin/sh
case "$1" in
  start)
tc qdisc add dev eth1 root handle 1: cbq bandwidth 10Mbit allot 9200 cell 16 avpkt 72 mpu 64
tc class add dev eth1 parent 1: classid 1:1 cbq bandwidth 10Mbit rate 10Mbit avpkt 72 prio 2 allot 9200 bounded
tc class add dev eth1 parent 1:1 classid 1:11 cbq bandwidth 10Mbit rate 64kbit avpkt 72 prio 2 allot 9200
tc class add dev eth1 parent 1:1 classid 1:12 cbq bandwidth 10Mbit rate 32kbit avpkt 72 prio 2 allot 9200
tc qdisc add dev eth1 parent 1:11 handle 11: tbf limit 20k rate 64kbit burst 1500 mtu 1500
tc qdisc add dev eth1 parent 1:12 handle 12: tbf limit 10k rate 32kbit burst 1500 mtu 1500
tc filter add dev eth1 parent 1: protocol ip prio 5 handle 1: u32 divisor 1
tc filter add dev eth1 parent 1: prio 5 u32 match ip tos 0x00 0xff flowid 1:12
tc filter add dev eth1 parent 1: prio 5 u32 match ip tos 0x20 0xff flowid 1:11
..
  stop)
tc filter del dev eth1 parent 1: prio 5
tc class del dev eth1 classid 1:12
tc class del dev eth1 classid 1:11
tc class del dev eth1 classid 1:1
tc qdisc del dev eth1 root
;;
```

6. TOKEN BUCKET FLOW PERFORMANCE

A set of experiments using the Token Bucket Flow queuing discipline was carried out with the purpose of evaluating the differences between the ideal shaping and the one offered in practice by the LINUX kernel. The physical configuration used for the tests consisted of two PCs directly connected on a 10 Mbit/s Ethernet. One PC generated traffic and the other received it and gathered statistics about the received traffic. The tool used for generating traffic was *rude 0.61* and, for gathering statistics, its counterpart *crude 0.61* was used. This tool allows creating UDP over IPv4 CBR flows with a good accuracy. To obtain the highest accuracy in the generated traffic

and measurements, the processes were run with maximum priority. The duration of the tests was always 60 seconds.

Application data is encapsulated with UDP and IP headers, which contribute with 28 octets of overhead per IP packet. Therefore, it is necessary to relate the data rate generated by the application with the raw IP rate. Since it is the latter that is controlled by the TBF discipline, it will be used when comparing the expected and the detected (measured) rates at the destination (the same relation would hold at application level).

In the tables of results, the labels have the following means: 1) Generated Data Rate represents a constant bit rate generated by the rude tool (data only); 2) Expected IP Rate is the expected raw IP rate at the detection side. This value is limited by the Token Bucket rate; 3) Detected IP Rate is the raw IP rate measured at the destination. The latter must be compared with the Expected IP Rate value and should not exceed it.

For a Token Bucket rate of 64 kbit/s and a datagram of 100 octets (data size – 72 octets), the results are shown in “Table 1”.

Table 1 - Packet size 100 octets, Token rate 64 kbit/s

Generated Data Rate	Expected IP Rate	Detected IP Rate
28 800	40 000	40 022
46 080	64 000	58 967
57 600	64 000	58 967

The critical data rate that corresponds to an IP rate equal to the Token Bucket rate is 46080 bit/s. When the data rate is below the critical value (first line), the detected rate is similar to the expected one. When the data rate is above that value (third line), the TBF controller is active; in this case the detected rate is below the expected one. The error is -7.9%. The second line, which corresponds to the critical value, exhibits the same behaviour.

The datagram size was then increased to 1228 octets (data size - 1200 octets) maintaining the same Token Bucket rate. The results are presented in “Table 2”.

Table 2 - Packet size 1228 octets; Token rate 64 kbit/s

Generated Data Rate	Expected IP Rate	Detected IP Rate
28 800	29 472	29 644
57 600	58 944	59 058
76 800	64 000	65 403
96 000	64 000	65 395

The critical data rate is now 62540 bit/s, which means that the last two lines correspond to an effective rate control.

The detected rates are now above the expected ones, but the errors are small. For example, considering the last line, the error is 2.2%. It seems that the packet size does have influence on the accuracy of the Token Bucket Flow and that there should be a packet size that optimises its performance.

The impact of Token Bucket rates on the performance of the TBF was evaluated, as well. Packet sizes of 100 and 1228 octets and Token Bucket rates of 8 and 128 kbit/s were tested. The results are shown in “Table 3”. In all cases the data rates are above the critical values and therefore the expected IP rates are the corresponding Token Bucket rates.

Table 3 - Packet size 100, 1228 octets; Token rate 8, 128 kbit/s

Generated Data Rate	Packet Size	Expected IP Rate	Detected IP Rate
8 000	100	8 000	7 477
8 000	1 228	8 000	8 301
128 000	100	128 000	117 778
128 000	1 228	128 000	130 610

Although absolute errors increase with the token rate, relative errors are similar. For a packet size of 100 octets, errors of -6.5%, -7.8 % and -8.0% were obtained for data rates of 8, 64 and 128 kbit/s, respectively. For a packet size of 1228 octets, errors of 3.8%, 2.2% and 2.0% were obtained for the same data rates. While with small packets, increasing the token rate increases the relative error, with larger packets, increasing the token rate decreases the relative error. For confirmation, another test, with a packet size of 1228 octets, was carried out with a Token Bucket rate of 1 Mbit/s. The detected IP rate for this configuration was 1 013 344 bit/s, which means a relative error of 1.33%, thus confirming the previous results.

Only one flow at a time was used in the previous tests. In a final test, two flows classified into two different classes, one for 64 kbit/s limit and the other for a 32 kbit/s limit were used. The results are shown in “Table 4”.

Table 4 - Two simultaneous flows:
Packet size 100 octets; Token rates 32 and 64 kbit/s

Flow	Token Bucket Rate	Detected IP Rate
1	32 000	29 567
2	64 000	58 967

The results for flow 2 are the same as those obtained on the first test. The existence of more than one Queuing Discipline/Class does not seem to affect the result. For flow 1, the relative error is –7,6 %, lower than for 64 kbit/s and higher than for 8 kbit/s, in conformance with what has been previously said about the effect of the token rate.

7. CONCLUSIONS

This paper describes the architecture of an UMTS terminal that supports IP based services and is implemented on a LINUX PC. The terminal is required to support 4 services (Videoconference, Video streaming, Web Browsing and Email) that are representative of UMTS traffic classes. Videoconference and Video streaming use the RTP/UDP/IP protocol stack and each service generates two real-time flows with QoS requirements. The other services are less QoS demanding and use the traditional TCP/IP stack. The UMTS protocol stack, implementing the Non-Access Stratum, is expected to be available as a module with two interfaces: the *umt0* network interface for IP packets (user plane) and a character device driver used to establish an terminate RABs (control plane).

The mapping of IP/RSVP flows into UMTS PDP contexts and the control plane in general are not considered. The paper mainly addresses the problem of classifying and shaping the packets passing from the IP layer to the UMTS interface so that the following goals are fulfilled: 1) packets are delivered in time and the application QoS requirements are satisfied but, on the other hand, 2) these flows do not violate the QoS contracts previously established between IP and NAS UMTS.

The solution proposed for this problem relies on the LINUX IP traffic control capabilities for classifying and shaping the flows – a CBQ queuing discipline is defined as a tree of classes, one for each RAB/PDP Context. Each leaf class is configured as a TBF queuing discipline that shapes the flow.

The tests carried out to validate the proposed solution show that errors between the ideal and the real shaping vary with packet size and token bucket rate in a range from 0 to 8%. These errors, however, seem to be constant and predictable. By biasing the token bucket rate, accurate results seem to be possible.

The main advantage of this solution is to rely on well known and widely available scheduling and shaping functions at IP level.

8. ACKNOWLEDGEMENTS

The authors wish to thank the support given by the IST research programme of the European Union and their partners within the ARROWS consortium: Universitat Politecnica de Catalunya, University of Limerick, Telefónica I+D and Telecom Italia Lab.

9. REFERENCES

[1] IST ARROWS project, http://www.arrows-ist.upc.es
[2] N.P. Magnani, F. Casadevall, A. Gelonch, S. McGrath, M. Ricardo, I. Berberana, “Overview of the ARROWS Project”, IST Mobile Communications Summit 2001, Barcelona, Spain, September 9-12, 2001.
[3] 3GPP TS 23.107 V5.0.0, “QoS Concept and Architecture”, April 2001.
[4] Paul White, “RSVP and Integrated Services in the Internet: A Tutorial”, IEEE Comm. Magazine, Vol. 35, No. 5, May 1997, pp. 100-106.
[5] 3GPP TS 23.207 V5.0.0, “End-to-end QoS Concept and Architecture”, June 2001.
[6] 3GPP TS 23.060 V5.0.0, “General Packet Radio Service (GPRS); Service description; Stage 2”, January 2002.
[7] iproute2+tc notes, http://snafu.freedom.org/linux2.2/iproute-notes.html
[8] Linux Iproute2, http://defiant.coinet.com/iproute2/
[9] Sally Floyd, "Notes on CBQ and Guaranteed Service", July1995.
[10] K. Dovrolis, M. Vedarn, P. Ramanathan, “The Selection of the Token Bucket Parameters in the IETF Guaranteed Service Class”, Technical Report, Department of ECE, University of Wisconsin-Madison, November 1997.