UMTS Terminal Equipment For All-IP Based Communication Scenarios

Manuel Ricardo[1,2], Rui Soares[1,2], Jaime Dias[2] and José Ruela[1,2]
{mricardo, rsoares, jdias, jruela}@inescporto.pt

1. Faculdade de Engenharia da Universidade do Porto. Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
2. INESC Porto. Praça da República 93, 4050-497 Porto, Portugal

***Abstract* - The paper presents the architecture of an UMTS terminal equipment for IP based communications, that supports relevant services and takes advantage of traffic control features available in LINUX.**

I. INTRODUCTION

This paper presents some results of the work carried out in the European IST ARROWS (Advanced Radio Resource Management for Wireless Services) project [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 of them based on IP. Therefore, the main objectives of ARROWS are: 1) to simulate and validate advanced RRM algorithms for an efficient use of the radio resources at UTRA; 2) to provide QoS bearer services for packet switched flows at the UTRA through the use of QoS management procedures; 3) to demonstrate the benefits of the developed algorithms and procedures by means of an IP based multimedia testbed.

The work presented in this paper is related to the third objective. This testbed consists of several functional blocks: 1) an all-IP based UMTS terminal; 2) an UTRAN emulator, implementing the UMTS radio interface and the relevant RNC functions; 3) a gateway implementing functions traditionally assigned to SSGN and GGSN; 4) a backbone IP network and its associated routers; 5) a server where applications have their servers.

In the remaining of the paper the first functional block will be addressed – the all-IP UMTS terminal, which supports multimedia applications and is implemented in a LINUX based PC. Its architecture, the applications selected, 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 organized in 6 parts. Section II introduces the multimedia applications selected for ARROWS and, thus, for terminal equipment. Section III presents the UMTS terminal architecture, which is biased towards the compatibility between the IP and UMTS worlds from flow and QoS points of view. Section IV reviews the IP QoS facilities nowadays available in Linux and presents a strategy for using them in an all-IP UMTS terminal. Section V describes the strategy adopted for controlling the traffic. Finally, Section VI presents the main conclusions.

II. MULTIMEDIA APPLICATIONS

In UMTS four traffic classes have been identified: conversational, streaming, interactive and background. The main distinguishing factor between these classes is how delay sensitive the traffic is: the conversational class is meant for very delay-sensitive traffic, while the background class is the least delay-sensitive. Furthermore, between the conversational and streaming classes, an additional comment can be made - the conversational class has more stringent jitter requisites. This is mostly due to the fact that conversational traffic is symmetric, while streaming is highly asymmetrical. This allows the use of buffers in streaming to smooth out jitter. In conversational services, this would increase too much the delay for human perception, turning the communication awkward.

In the ARROWS project, one application representative of each UMTS traffic class was selected: Videoconference (conversational), Video streaming (streaming), Web browsing (interactive) and Email (background). Besides, applications were required to satisfy three characteristics: 1) be widely used; 2) be open source, so that extensions to RSVP and IPv6, for instance, could be easy; 3) have port for LINUX.

A. Videoconference

VIC and RAT, the well-known video and audio conferencing tools, were selected as departing applications. Although designed for multicast environments, they will be configured in ARROWS as point-to-point (unicast). Both applications rely on the Real Time Transport Protocol (RTP).

RTP is the IETF protocol for the transport of real-time data, including audio and video. It can be used for media-on-demand as well as interactive services such as Internet telephony. RTP consists of a data and a control part. The latter is called RTCP. The data part of RTP is a thin protocol providing support for applications with real-time properties such as continuous media (e.g., audio and video), including timing reconstruction, loss detection, security and content identification. RTCP provides support for real-time conferencing of groups of any size within an internet. It offers quality of service feedback from receivers to the multicast group as well as support for the synchronization of different media streams. Both protocols use the services provided by the UDP protocol that, in turn, uses IP.

VIC supports the H.261 and H.263 video codecs. H.261 is a video coding standard designed for data rates which are multiple of 64 kbit/s and is sometimes called px64 kbit/s (p in 1 to 30). H.263 was designed for low bitrate communication (less than 64 kbit/s). RAT also supports several codecs, among which the G.711 PCM

(64 kbit/s), the G.726 ADPCM (16-40 kbit/s), the LPC (5.6 kbit/s) and the 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 (RAB – Radio Access Bearers / PDP Contexts).

B. Video streaming

Video streaming (including audio, as well) was required to be based on a coding scheme using two streams that contain base and enhancement information, respectively. The intention was to adopt MPEG-4 FGS (Fine-Granularity Scalability) profile. This codec, however, is under development and some parts (the scalable parts) are not yet stable. To overcome this limitation, an H.263+ video codec will be used instead. It can generate two constant bit rate flows – a 32 kbit/s base flow and an enhancement flow that can vary from 0 to 96 kbit/s in steps of 16 kbit/s.

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 film, two unidirectional and real time IP flows have to be transported over the UMTS network.

C. Web Browsing and Email

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.

III. TERMINAL ARCHITECTURE

The UMTS terminal architecture for supporting the IP based services presented above is shown in Fig. 1.

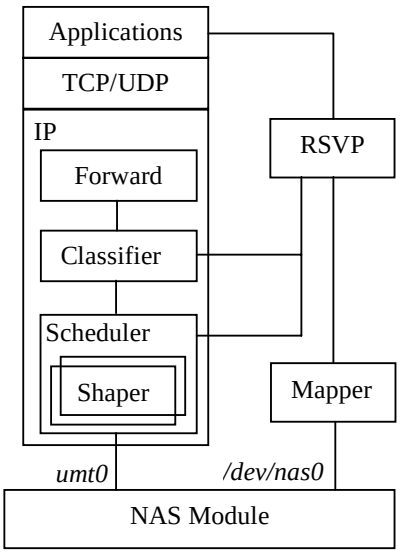

Figure 1 – UMTS Terminal Architecture

A. 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. Datagram fields, such as destination port, TOS (IPv4) or Flow Label (IPv6), 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 its bandwidth. It can shape a flow so that it does not violate the QoS previously negotiated for the RAB (in the NAS module) sustaining it.

Mapper is the block responsible for negotiating the activation, modification and deactivation of PDP contexts associated with the RABs [4], passing the NAS the desired QoS parameters. Also, the mapping between RSVP QoS parameters and PDP context QoS parameters is performed here. Finally, this block implements the admission control required for RSVP.

RSVP implements the RSVP protocol that, in the ARROWS project, is used to guarantee end-to-end QoS to IP flows transport through both the UMTS and the IP backbone. It can, in some circumstances, be avoided.

NAS module implements the 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 Fig. 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 Fig. 1), through which messages for establishing and terminating RABs are exchanged.

B. PDP context

A GPRS subscription consists of one (or more) *PDP address* that, in the case of Fig. 1, will be the IP address associated to the interface *umt0*. Each PDP address, in GPRS, is described by one or more *PDP contexts*. 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 (TOS) (IPv4) / Traffic Class (IPv6) and Mask, Flow Label (IPv6). PDP context and RAB, due tot heir 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 the IP packets into the appropriate RAB. This justifies the use of the *Classifier* and the *Shaper* in Fig. 1. With this, more than one PDP context may exist, each with different QoS parameters and to which packets will be forwarded depending on the class of the traffic they belong. 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.

C. RSVP

Applications will interface with *RSVP* to request the reservation of resources along the path of the data stream, at the IP layer. This block will also interface with the *Mapper* so that the establishment and release of a PDP context may be done. In the case of activation, it should map the RSVP QoS parameters into appropriate PDP context QoS parameters and ask the NAS for the establishment of a PDP context with the given parameters. It should be noted that this block works also as admission control for the RSVP.

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

IV. IP TRAFFIC CONTROL IN LINUX

Recent LINUX kernels have been growing to include a number of advanced networking features such as firewalls, QoS and tunneling. 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 Fig. 2.

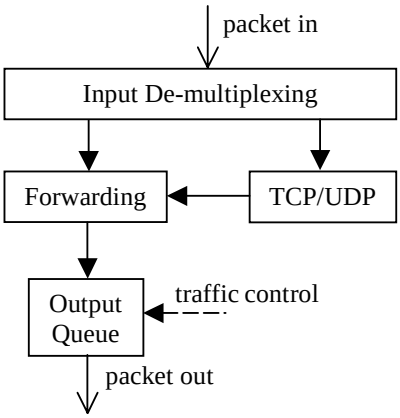

Figure 2 – LINUX traffic control

The *Input De-multiplexing* block examines an incoming packet and determines if it is destined 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. After this, the packet is placed in a queue maintained for the device (e.g. *eth0* or *umt0*). It is on this *Output Queue* that traffic control is performed, 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 Fig. 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*, however, may be structurally complex. In this case, 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 a new *Queuing Discipline* associated that, in turn, may have more *Classes* and *Filters*. This principle enables 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 enables the classification of packets based on any of their header fields, such as IPv4, IPv6, TCP, UDP or ICMP. 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 [2] and [3].

Some relevant header files associated to traffic control are the *include/net/pkt_sched.h* and the *include/linux/skbuff.h*. The last includes, among others, the definition of an important structure – the *sk_buff*. Every IP datagram in the kernel has associated one instance of this structure that contains information such as: the socket the packet belongs to (*struct sock *sk*), the device the packet has arrived on or is leaving from (*struct net_device *dev*), the transport header (*union h*), the network header (*union nh*), the link layer header (*union mac*), the packet queuing priority (*__u32 priority*), the traffic control index (*__u32 tc_index*) and the data itself (*uchar *data*). One important function implementing traffic control, the *int (*enqueue)(struct sk_buff *, struct Qdisc *)*, is responsible for queuing the packet pointed to by the first parameter, with the queuing discipline pointed to by the second. When the function is executed, the filters are run one by one until a match occurs (in the case where filters apply). This determines the class the packet belongs to. After that, the *enqueue* function of the queuing discipline associated with that class is called and so on. The *enqueue* function for the Token Bucket Flow queuing discipline, for instance, is *tbf_enqueue* and is defined in *net/sched/sch_tbf.c*.

V. 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.

A. Scheduling

Scheduling is usually required when there is the need to share a link with limited bandwidth. Also, 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 talking about 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 Sec. II can, from the priority point of view, be ordered as Email, Web Browsing, Video streaming and Videoconference, where the last has the highest priority. The same, however, may not be said about bandwidth. A Video streaming session may require a larger bandwidth than a pure voice session.

The CBQ (Class Based Queuing) discipline can be used to solve the priority issue. This discipline is based on a statistical scheduling and on 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 aims at guaranteeing a low delay to real time flows. The link sharing scheduler tries to avoid that real time flows monopolize the use of the link.

B. Shaping

The bit rate of a flow can be regulated using shaping techniques. In this case, the traffic passed to a RAB needs to be in conformity with 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, in terms of the difference between the configured parameters and offered results, can be achieved if a TBF (Token Bucket Flow) queuing discipline is associated to each leaf class.

The TBF consists of a buffer (bucket), filled by 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 rate of incoming tokens. In this case each incoming packet has its matching tokens and passes the queue without delay; 2) the data arrives into TBF at a rate lower than the token rate. In this case, only some tokens are deleted when data packets are sent out and therefore tokens accumulate up to the bucket size. These tokens can then be used to send data above the token rate, if short data burst occurs; 3) the data arrives into TBF at a rate higher than the token rate. In this case, a flow overrun occurs - incoming data can be only sent out without loss until all accumulated tokens are used. After that, data packets are dropped. This scenario enables data to be shaped administratively. The accumulation of tokens allows short burst of data to be passed without loss, but any lasting overload will cause packets to be constantly dropped. The average data rate is bounded by the token rate.

C. Proposed configuration

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

For generality and to prove the flexibility of the solution, four leaf classes were drawn. In order to support the services presented in Sec. II, a finer assignment is required. Videoconference requires two classes, corresponding to two RABs and serving two flows - one for audio and another for video. Video streaming also requires two classes – for the base and the enhancement flows. The other flows, corresponding to Web Browsing, Email and generic signaling, 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.

There is one problem associated to this solution - 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 (IPv6 case) or the TOS (IPv4) fields to distinguish the RABs. The TFT (Traffic Flow Template, 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.

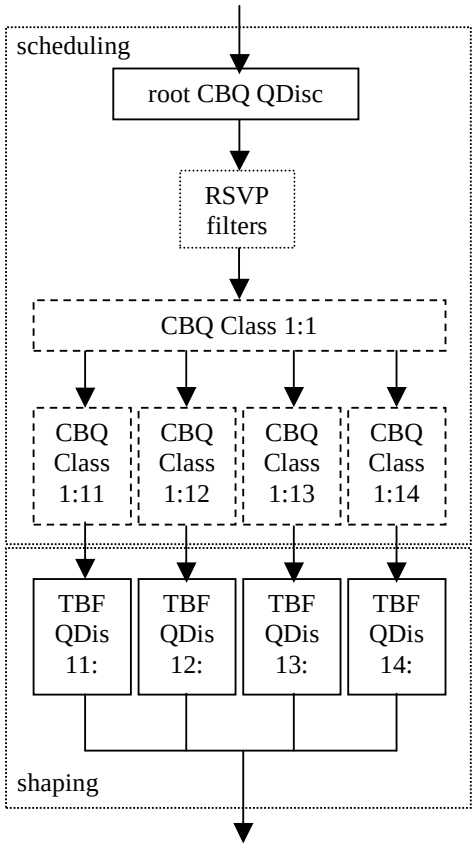

Figure 3 – UMTS terminal traffic control

However, a RAB may have more than one TOS or Flow Label associated with it (up to eight in conformity with 3GPP standards). This requires that applications mark the flow label or TOS field. This can be done in two ways.

For the case of *TOS*, the following function can be used by applications

setsockopt(send_sock, SOL_IP, IP_TOS, &tos, sizeof(tos))

where *send_sock* is the socket descriptor and *tos* is a variable with the TOS value. This call can be done after opening the socket. From this point on, all packets belonging to this socket have the TOS field marked with the desired value.

For the *Flow Label* case, the solution is more complex. First, the kernel must be requested to lease the flow label and then the flow label with the value leased must be set. The following (simplified) function performs that:

```
int get_flow_label(int fd, struct sockaddr_in6 *dst, __u32 fl)
{
int on = 1;      struct in6_flowlabel_req  freq;
freq.flr_label = htonl(fl);      freq.flr_action = IPV6_FL_A_GET;
freq.flr_flags = IPV6_FL_F_CREATE | IPV6_FL_F_EXCL;
freq.flr_share = IPV6_FL_S_EXCL;
memcpy(&freq.flr_dst, &dst->sin6_addr, 16);
setsockopt(fd, SOL_IPV6, IPV6_FLOWLABEL_MGR, &freq,sizeof(freq));
dst->sin6_flowinfo |= freq.flr_label;
setsockopt(fd, SOL_IPV6, IPV6_FLOWINFO_SEND, &on, sizeof(on));
return 0;
}
```

This function must be called prior to calling the socket connect function. In fact, a socket connect MUST be called even for UDP flows, otherwise the flow label will not be marked. The parameters

passed are the socket descriptor, *the struct sockaddr_in6* used on the connect call and the desired flow label. Also, after leased by the kernel, the flow label cannot be used in any other socket. This is given by line *freq.flr_share = IPV6_FL_S_EXCL*; other flags may be used to set the flow label share options.

Marking the TOS or Flow Label at application level implies that applications must be changed. An alternative approach is to mark the packet when it is *enqueued*, on the queuing discipline. In this case, each RAB would have a TOS or Flow Label associated with it.

This solution requires patching the kernel, adding the following line on the *tbf_enqueue* function on *net/sched/sch_tbf.c*:

```
tbf_enqueue(struct sk_buff *skb, struct Qdisc* sch) {
  struct tbf_sched_data *q = (struct tbf_sched_data *)sch->data;
  skb->nh.iph->tos = 0xAA;
  …
```

This marks the packet TOS with the value *0xAA*. Obviously, the value would have to be configured for each queuing discipline. This could require a user level tool to tell the kernel which value to use.

D. 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 Fig. 3, but for device *eth1*, is shown below.

Only two classes are configured, with token rates of 64 kbit/s 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 kbit/s 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
;;
```

VI. CONCLUSIONS

This paper reports some results of the work carried out within the European R&D project ARROWS and addresses 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 relevant UMTS traffic classes. Videoconference and Video streaming use the RTP/UDP/IP protocol stack and each service generate 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 well-known Non-Access Stratum, is expected to be available as a module offered 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 [4] are not considered and the paper mainly addresses the problem of classifying and shaping the packets passing from the IP layer to the UMTS interface so that (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 is to rely 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 paper also reasons about the proposed solution by identifying the necessity of marking the IP packets so that they can be selected by *rsvp* filters (that assign packets to the classes) but can also be parsed by the UMTS network interface (that uses this information to assign packets to RABs). The main advantage of the solution [5] is to rely on well-known and widely available scheduling and shaping functions at the IP level.

The paper gives examples for configuring the traffic control and for marking packets in LINUX, as well.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] IST ARROWS project, http://www.arrows-ist.upc.es

[2] iproute2+tc notes, http://defiant.coinet.com/iproute2/, http://snafu.freedom.org/linux2.2/iproute-notes.html

[3] Rui Prior, “Quality of Service in Packet Switched Networks” (in Portuguese), MSc Thesis, Univ. Porto, 2001. http://telecom.inescn.pt/doc/msc.html

[4] R. Soares, M. Ricardo, “Description of the Interface between the IP and UMTS communication layers (signalling and user planes), at the User Terminal and Gateway equipment”, ARROWS document, 2001.

[5] R. Soares, M. Ricardo, “Traffic Control in the UMTS Terminal based on IP CBQ and TBF queuing disciplines: description of available implementations for Linux, proposal of a traffic control architecture mappable to UMTS PDP contexts and flow shaping evaluation”, ARROWS document, 2001.