Dynamic resource configuration in DiffServ network: control plane mechanisms and performance evaluation of a Traffic Control API

S. Giordano[1], M. Listanti[2], F. Mustacchio[1], S. Niccolini[1], S. Salsano[3], L. Veltri[4]

[1] Dept. of Information Engineering – University of Pisa, Italy
{s.giordano,fabio.mustacchio,s.niccolini}@iet.unipi.it
[2] INFOCOM – University of Rome "La Sapienza", Italy
listanti@infocom.uniroma1.it
[3] DIE – University of Rome "Tor Vergata", Italy
stefano.salsano@uniroma2.it
[4] Dept. of Information Engineering – University of Parma, Italy
luca.veltri@unipr.it

Abstract. Voice, video and multimedia applications are sensitive to the QoS provided by the underlying IP network. The DiffServ architecture offers a set of QoS mechanisms for IP networks. The "binding" of the applications QoS needs with the QoS features offered by the DiffServ networks is still an open problem. The simplest approach is to have a static configuration of QoS and therefore no direct interaction in the control plane between applications and QoS. We consider the advanced scenario, where the QoS mechanisms can be dynamically configured to follow the applications' need. For this scenario, a set of control plane interfaces needed for a whole end-to-end QoS architecture is defined. At the lower lever, an internal interface ("Application Programming Interface" – API) in the QoS router is considered. This interface provides access to the DiffServ QoS mechanism available in a router and it is used by the control logic running in the router itself. Then a QoS signaling protocol is considered, that allows external QoS clients to dynamically access the QoS services provided by the network. Finally the interaction of a session level signaling protocol (i.e. the SIP for IP telephony) with the QoS protocol is defined. The testbed implementation of proposed architecture and a set of performance tests on the internal QoS API are reported.

1. Introduction

The automation of the resource allocation process and of network element configuration within a QoS network is a hot topic in the Internet community. Looking at the standardization effort in the area of

IP QoS the two main approaches that have been proposed in the IETF are the Integrated Services (IntServ) model and the Differentiated Services (DiffServ) model. Additional proposals consider a combination of the two approaches. In addition, the MPLS (Multi Protocol Label Switching) technology is going to play an important role in this field, for example as transport backbone for DiffServ. A very good introduction to IP QoS topics can be found in [1], [2].

The DiffServ architecture has the potentiality, with its Per Hop Behaviors (PHBs), to differentiate the QoS in a scalable mode; unfortunately such architecture is still utilized in a static manner. The providers are configuring the network resources statically with a capacity planning study without a time-dependent optimization. Since the amount of traffic offered to the network is intrinsically variable with time there is the possibility of underutilizing the resources or overloading the network.

The IntServ architecture, on the other hand, is a more dynamic one and it is more suited to address time-variant configuration issues but it has shown its weakness when dealing with scalability problems.

In the current DiffServ framework, even if is possible to define a static SLA (Service Level Agreement), it is difficult for a client to rely on such SLAs for several reasons:

- the maintenance of the information about all the users introduces scalability issues;
- the user could change the call parameters during the call, paying for unused resource;
- it is difficult to define static SLAs that well suite to the user demand.

For these reasons dynamic resource allocation in a QoS domain is a very important issue.

In particular, this work is focused on providing a scalable architecture for dynamic resource allocation / configuration when an access network (which may be QoS aware or not) requests transport to a "*QoS enabled*" network. The proposed QoS architecture is based on two important aspects: the DiffServ based resource management scheme for guaranteeing a suitable level of QoS, and the signaling mechanisms for providing dynamic resource reservation. The first aspect mainly deals with "Traffic Control" (TC) and scheduling strategies, while the latter deals with signaling protocols.

Traffic Control is the basic element of the QoS architecture, implemented in the QoS capable routers. Such QoS router should classify QoS enabled packets and handle the packets applying a proper forwarding treatment. These Traffic Control capabilities must be configured dynamically, i.e. the

routers should offer an interface to accept configuration requests related to the setup and release of QoS flows. Traffic Control capability and interface aspects will be analyzed in this work.

Regarding the signaling mechanisms, different solution have currently been considered by standardization groups. Since QoS aspects are particularly important for real-time multimedia applications (e.g. IP Telephony, Videoconferencing, etc.), we will analyzed a dynamic reservation architecture based on the SIP (Session Initiation Protocol) protocol as application level protocol. SIP [3] is currently having a lot of attention within IETF (Internet Engineering Task Force) and it seems to be the more promising candidate as call setup signaling for the present day and future IP based telephony services (e.g. usage of SIP in the Internet Multimedia Subsystem (IMS) specifications is chosen by 3GPP group [4]). It could even be a real competitor to the Plain Old Telephone Service (PSTN). For the realization of this scenario, there is the obvious need to provide a good speech quality. This quality in turn depends on the Quality of Service delivered by the IP network. Reservation and/or admission control mechanisms could be needed to get QoS from the IP network. Unfortunately, at present day, there is not a clear picture about the "elected" mechanism for QoS provisioning in IP network, as much research and standardization effort is ongoing in this area. The interaction of these QoS mechanisms with the call setup procedures (i.e. SIP) is therefore a very hot topic. There is a recent work of the SIP IETF Working Group [5], which deals with the interaction between SIP and resource management for QoS.

Our goal is to enable seamless inter-operation between the application level protocol, the policy/resource management protocol and the router configuration. To accommodate this aim we propose, in this work, a very simple solution that is based on an enhancement of the SIP protocol to convey QoS related information. The solution preserves backward compatibility with current SIP applications and it de-couples as much as possible the SIP signaling from the handling of QoS. Moreover the solution foresees the use of COPS (extended with QoS handler features) protocol as policy/bandwidth control and the use of Linux Traffic Control (TC) to build the QoS mechanisms.

The rest of the paper is organized as follows. Section 2 gives an overview of the proposed mechanisms for dynamic resource allocation explaining the rationale and overall requirements. Those mechanisms are detailed in Sections 3 and Section 4, dealing with Traffic Control and QoS signaling mechanisms, respectively. In Section 5 a test-bed implementation of the overall architecture is

described, while Section 6 reports the test results on the performance of the Traffic Control API. Finally, in Section 7, we give our conclusions.

2. Dynamic Resource Configuration: Mechanisms Overview

The configuration of the DiffServ Traffic Control mechanisms in the routers can be "triggered" in several ways. The simplest approach, suitable for a "static" QoS configuration is to manually operate with the router Command Line Interface (CLI). More sophisticated solutions, that always operate on the management plane (i.e. suitable for static QoS), are based on management protocols like SNMP [18] or COPS-PR [6]. Coming to a dynamic approach based on signaling, the IntServ architecture dictated the use of the RSVP protocol. We will describe the use of a variant of the COPS [7] protocol, named COPS-DRA (Dynamic Resource Allocation) [8] as the QoS signaling protocols that triggers the resource configuration in the routers. Fig. 1 provides a representation of the different resource configuration mechanisms of a DiffServ router.

Internally the routers will have a primitive interface which will handle the configuration commands (coming from CLI interface, management or signaling protocols) translating into directives on the router hardware. This interface is typically not available in a commercial router. In case of the software router based on the open source Linux Operating System, such interface is available and it is called "TC API" [15]. The TC API allows a controlling entity to configure the traffic control modules (classifier, policer, marker, queuing disciplines) according to the dynamic setup and release of QoS flows. In dynamic approach considered in this work the controlling entity resides in the router and it is in turn controlled by means of a QoS signaling protocol. The performance of the dynamic configuration mechanisms in the routers may constitute a bottleneck in a dynamic QoS architecture. For example, a dynamic configuration mechanism based on Command Line Interface could not provide adequate performance in most current commercial routers. In Section 6 we will present some simple "black-box" measurements related to the TC API performance in a Linux based software router.

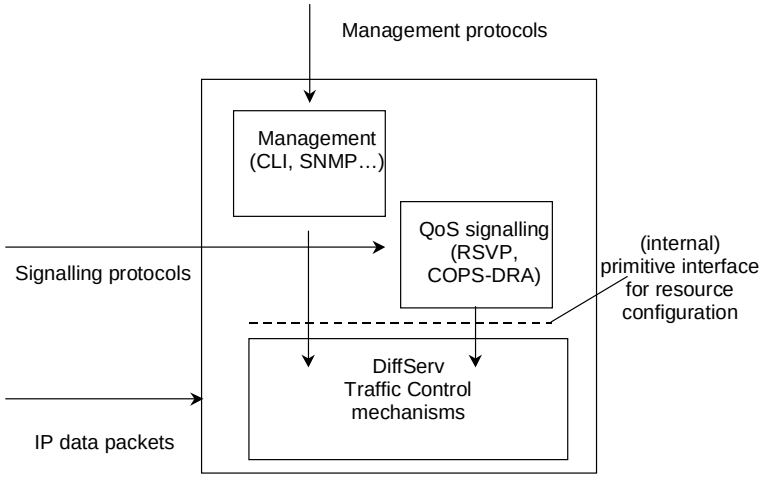

Fig. 1. Resource configuration mechanisms in a DiffServ router

The IntServ "traditional" RSVP approach foresees that QoS signaling is initiated by the end terminals and strictly follows the data path. Therefore each router in the path receives resource reservations, it can take local admission control decisions and then it can configure the Traffic Control mechanism. In a Linux software router this last step means interacting with the TC API.

In the proposed architecture a "QoS provider" is placed in each Edge Router (ER). The COPS-DRA protocol is used by a "QoS client" to make reservation requests to the QoS provider for admission in a QoS enabled network. The QoS provider takes an admission control decision and then it uses the TC API to configure the Traffic Control mechanisms. Differently from basic RSVP approach, the admission control decision taken by the QoS provider in the Edge Router does not only refer to the local link, but it can be related to an edge-to-edge path in the QoS enabled network. For this reason, in the resource management and admission control process the Edge Routers may be supported by a logically centralized entity, called Bandwidth Broker (BB). The signaling dialogue between Edge Routers and Bandwidth Broker is based on COPS-DRA as well.

Another important difference from basic RSVP approach is that the COPS-DRA signaling does not need to strictly follow the data path and the signaling does not need to be initiated by the end terminal. Therefore the user terminal is not "forced" to be aware of the network QoS model, and it should not originate the resource reservation with an "ad hoc" protocol (e.g. RSVP). In our belief such a solution limits the scalability of the architecture and needs a capillary control on every user terminal.

However triggering the resource reservation is mandatory in order to take advantage from the flexibility of a dynamic Service Level Agreement (SLA) scheme. The proposed solution is to enhance the session initiation signaling in order to care for QoS. In particular we considered the use of the SIP protocol in order to trigger the resource reservation.

The solution foresees the enhancement of the SIP proxy servers to handle QoS aspects. In the following, the enhanced SIP server will be called Q-SIP server (QoS enabled SIP server). All the QoS aspects can be covered by the Q-SIP servers in the originating and terminating sides, no involvement of the terminals is needed, therefore the user terminal can be decoupled by the knowledge of the QoS model used in the network. Note also that in a DiffServ QoS scenario there will be servers dedicated to policy control, accounting and billing aspects. Hence, a solution based on SIP servers is really suited to this QoS scenario.

Once the resource reservation is accepted the configuration phase takes place by means of kernel configuration. In our solution a Traffic Control Server is used configure the Linux kernel; it takes the input from the COPS-DRA client located in the Edge Router and configures the DiffServ classes. Once the reservation process ends the Linux PCs are properly configured in order to forward the user packets with the requested QoS (the necessary parameters are extracted from the Session Description Protocol, SDP, in the SIP messages).

3. Traffic Control and its interaction with signaling

As discussed in the previous section, the Traffic Control (TC) functionality of a router can be controlled by means of several mechanisms. We define a TC server which implements a “TC server API” and offers services to a controlling entity. In our architecture the controlling entity will manage the QoS Signaling Protocol (COPS-DRA) receiving requests coming from the QoS client (see Fig. 6). The choice to build a TC server as a distinct module preserves the modularity of the architecture. The TC server is a flexible element that could work also with different resource allocation mechanisms (e.g. SNMP, RSVP, etc.).

Let us consider how the resource allocation is performed within the QoS–enabled routers. We consider open-source Linux routers where the kernel provides the Traffic Control software functions allowing the building of a DiffServ router. The component of the Linux operating system responsible

for both bandwidth sharing and packet scheduling is called Traffic Control (TC) [14]. The main goal of the Linux Traffic Control is to manage packets queued on the outgoing interface with respect to the configured rules for the outgoing traffic. To achieve this goal the components used are: queuing disciplines, classes, filters and policing functions.

Filters are needed to mark packets on the ingress interface with a *tcindex* value (a kernel level packet specific value); it is used to forward the packets properly on the egress interface. Filtering rules could be performed on the following fields of the TCP/IP header: IP source address, IP destination address, source port, destination port, protocol, type of service. Policing functions are used in order to keep the traffic profile under the negotiated SLA level. Queuing disciplines and classes allow to implement PHBs in the DiffServ network.

The module responsible for resource allocation is named TC server and is located in the Edge Router. The COPS-DRA client, located in the Edge Router itself, acting as a TC client issues commands on the TC Server using the "TC Server API".

Communications between TC server and client on the TC server API use a simple proprietary protocol. The messages sent are TLV (Type-Length-Value) messages. The design foresees five message types:

- *REQUEST* (to set-up filter and bandwidth allocation for a given flow);
- *RELEASE* (to delete structures set by the *REQUEST* message);
- *MODIFY* (to modify the bandwidth allocation for a specific flow);
- *ACK* (to notify the correct reception of the message);
- *NACK* (to notify an error in the reception).

REQUEST messages brings information about the flow identifier, the service class, the amount of reservation and the filter set-up information.

The operational model of this message exchange is the following: when the controlling entity (TC client) decides that a QoS flow must be configured, it sends a *REQUEST* message to the TC server to configure the kernel parameters negotiated by means of COPS-DRA protocol. Then, TC server replies to the TC client with an *ACK* or *NACK* message to notify whether the message is received correctly or not. The operational model is the same for the *RELEASE* and the *MODIFY* messages.

Moreover, in order to achieve higher scalability, we implemented a second operational model where the resources are allocated in advance, up to a configurable quantity, by the COPS server running on the Bandwidth Broker. When the Bandwidth broker instructs the controlling entities by means of the COPS-DRA protocol of the advance reservation, the controlling entities sends a *REQUEST* message to their TC servers to configure in advance the kernel parameters. Afterwards, only when new requests make the resources becoming insufficient, a *MODIFY* message is sent to the interested TC server thus modifying the original allocation. For sake of scalability and efficiency of operations a combination of the two operational models is adopted in our solution.

4. COPS-DRA and Q-SIP signaling mechanism

In this section we first describe the proposed mechanism for admission control in a DiffServ network, named COPS-DRA (COPS-DiffServ Resource Allocation). Then we define the QoS mechanisms in the SIP protocol and its interaction with the COPS-DRA mechanism.

The basic idea is that Admission Control entities running on the network borders (e.g. in the Edge Routers) dialogues with external QoS clients (the Q-SIP servers) and with Bandwidth Broker in the "*QoS enabled"* network. The Admission Control entities use the COPS-DRA protocol to dialogue with the QoS clients and with the Bandwidth Broker.

The COPS (Common Open Policy Service) protocol is a simple query and response protocol that allows policy servers (PDPs, Policy Decision Points) to communicate policy decisions to network devices (PEP, Policy Enforcement Point). "Request" messages (*REQ*) are sent by the PEP to the PDP and "Decision" (*DEC*) messages are sent by the PDP to the PEP. In order to be flexible, the COPS protocol has been designed to support multiple types of policy clients. We have defined the COPS-DRA client type to support dynamic resource allocation in a DiffServ network.

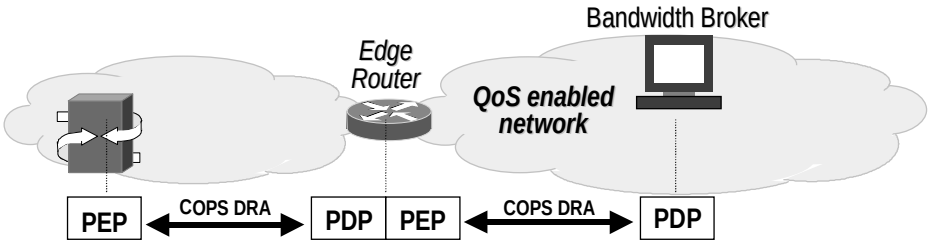

Fig. 2. COPS support to dynamic DiffServ based IP QoS

As a generic example, in Fig. 2 it is depicted a representation of the proposed architecture for dynamic DiffServ QoS. The COPS protocol is used on both the interface between the QoS client and the network, and the interface between the Edge Router and the logically centralized admission / policy control server. In Fig. 2 the QoS client is represented by a server for IP telephony and the leftmost interface is a User-to-Network interface. The architecture can easily support other scenarios where the QoS client belongs to the provider network (for example a SIP server in a 3rd generation mobile network).

With respect to the generic example, in Fig. 3, the COPS-DRA protocol is used on two different interfaces.

First, it is used as a generic signaling mechanism between the user of a "*QoS enabled*" network and the QoS provider. In our case the Q-SIP proxy server plays the role of QoS user and will implement a COPS-DRA client, while the Edge Router plays the role of QoS provider and will implement a COPS-DRA server. On this interface, COPS-DRA provides the means to transport: the scope and amount of reservation, the type of requested service and the flow identification. The second interface where the COPS-DRA is applied is between the Edge Router and the Bandwidth Broker, in order to perform the resource allocation procedures. A flexible and scalable model for resource allocation is implemented. A set of resources can be allocated in advance by the Bandwidth Broker to the Edge Router in order to accommodate future request (according to the so-called COPS *provisioning* model). The amount of this "aggregated" allocation can also be modified with time. Moreover, specific requests can be sent by the Edge Router to allocate resources for a given flow (according to the so-called COPS *outsourcing* model). The set of Edge Routers and the Bandwidth Broker realize a sort of distributed bandwidth broker in a DiffServ network. The generic COPS-DRA architecture is better described in [9], the protocol details can be found in [8].

By means of the COPS-DRA protocol, the Edge Routers provide a generic mechanisms that can be used by different QoS clients. In this paper we describe how the SIP servers may use the COPS-DRA in order to provide QoS to real-time applications (e.g. telephony, videoconferencing…). The whole end-to-end scenario is depicted in Fig. 3.

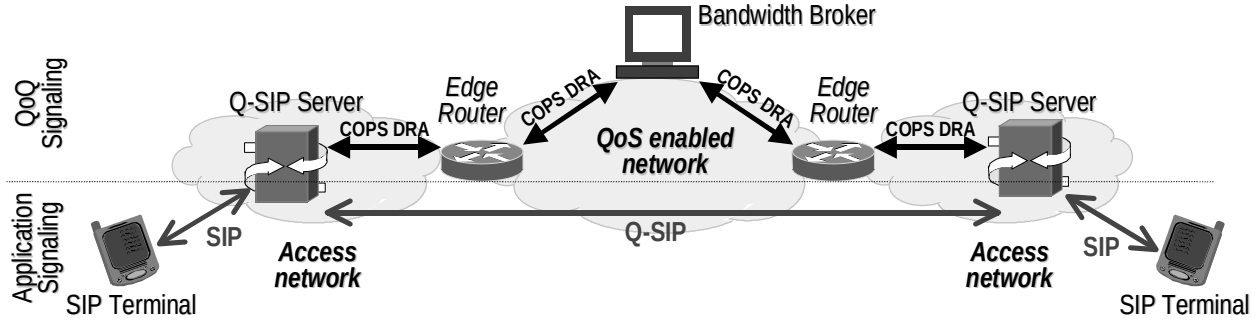

Fig. 3. End-to-end QoS scenario for the proposed architecture

The SIP signaling protocol needs to be extended in order to support a QoS enabled resource reservation. The extended version of the protocol will be called Q-SIP [10], [11]. The SIP proxy servers that handle the QoS extensions are called Q-SIP proxy servers. When needed, these servers originate the resource reservation requests to the DiffServ network (to the local or to the remote Bandwidth Broker as needed), by means of COPS-DRA protocol. The managing of such information is done at application level by means of proper coordination of Q-SIP and COPS-DRA protocols: in order to establish the correct resource reservation and configuration the Q-SIP servers interact with a COPS-DRA client. The first COPS-DRA client encountered in the resource reservation process is located in the same machine as the Q-SIP proxy server. The COPS-DRA client asks for the resource reservation to a COPS-DRA server located in the Edge Router of the DiffServ network. The Edge Router in turn may ask the remote Bandwidth Broker for resource. The decision whether to ask for resources to the remote Bandwidth Broker is taken with attention to the local resource availability; a mix of the so-called COPS configuration and outsourcing model is preferred in order to take advantage from the scalability of the former and the high control provided by the latter.

The proposed solution makes possible to use existing SIP clients with no enhancements or modifications. It is possible to interact with no problem with other parties that do not intend or are not able to use QoS. Moreover backward compatibility with standardized SIP protocol is preserved.

The IP phones/terminals are located on the access networks; standard SIP clients can be used, set with an explicit SIP proxying configuration. When a call setup is initiated, the caller SIP client starts a SIP call session through the SIP proxy server. If a Q-SIP server is encountered, this can start a QoS session interacting with a remote Q-SIP server and with the QoS providers for the backbone network (i.e. the access ERs). Fig. 3 shows the reference architecture.

According to the direction of the call, the two Q-SIP servers are named caller-side Q-SIP server and callee-side Q-SIP server.

As far as the reservation procedure is concerned, two different models are possible: i) unidirectional reservations and ii) bi-directional reservations. The choice between the two models can be done on the basis of a pre-configured mode or through the exchange of specific parameters (*qos-mode* parameters) between the Q-SIP servers during the call setup phase. We are now going to detail the uni-directional model because the bi-directional one is easily evinced from the previous.

In Fig. 4 we provide an example of the message exchange between Q-SIP servers, Edge Routers and Bandwidth Broker. The call setup starts with a standard SIP *INVITE* message sent by the caller to the local Q-SIP server (i.e. caller-side Q-SIP server). The message carries the callee URI in the SIP header and the session specification within the body Session Description Protocol (SDP) (media, codecs, source ports, etc). The Q-SIP server is seen by the caller as a standard SIP proxy server. The Q-SIP server, based on the caller id and on session information, decides whether a QoS session has to be started or not. If a QoS session is required/opportune, the server inserts the necessary descriptors within the *INVITE* message and forwards it towards the callee. The *INVITE* messages can be relayed by both standard SIP proxy servers and Q-SIP servers until they reach the callee-side Q-SIP server and then the invited callee. When the callee responds with a *200 OK* message, it is passed back to the callee-side Q-SIP server. At this point, the callee-side Q-SIP server can request a QoS reservation to the Edge Router on the callee access network (i.e. the QoS provider for the callee). Subsequently, the *200 OK* response, opportunely extended by the callee-side Q-SIP server, is forwarded back to the caller, via standard SIP servers and via the caller-side Q-SIP server. When the caller-side Q-SIP server receives the *200 OK* message, it performs QoS reservation with the Edge Router on the caller access network (i.e. the QoS provider for the caller).

As for the COPS-DRA messages, The first message is originated by the COPS-DRA Client co-located with Callee Q-SIP Server once it receives the *200 OK* message from the Callee SIP Terminal.

The dotted lines in Fig. 4 are optional messages needed if the outsourcing model is adopted and an outsourced *REQ* message is sent to the BB. Once the *200 OK* message arrives to the Caller Q-SIP Server an analogous message exchange (*REQ-DEC*) is performed (since we are detailing an uni-directional model).

It is important to note that the proposed architecture keeps the compatibility with standard SIP clients and standard SIP servers. All the information needed by the Q-SIP servers to perform the QoS session setup is inserted within the SIP messages in such a way that non Q-SIP aware agents can transparently manage the messages.

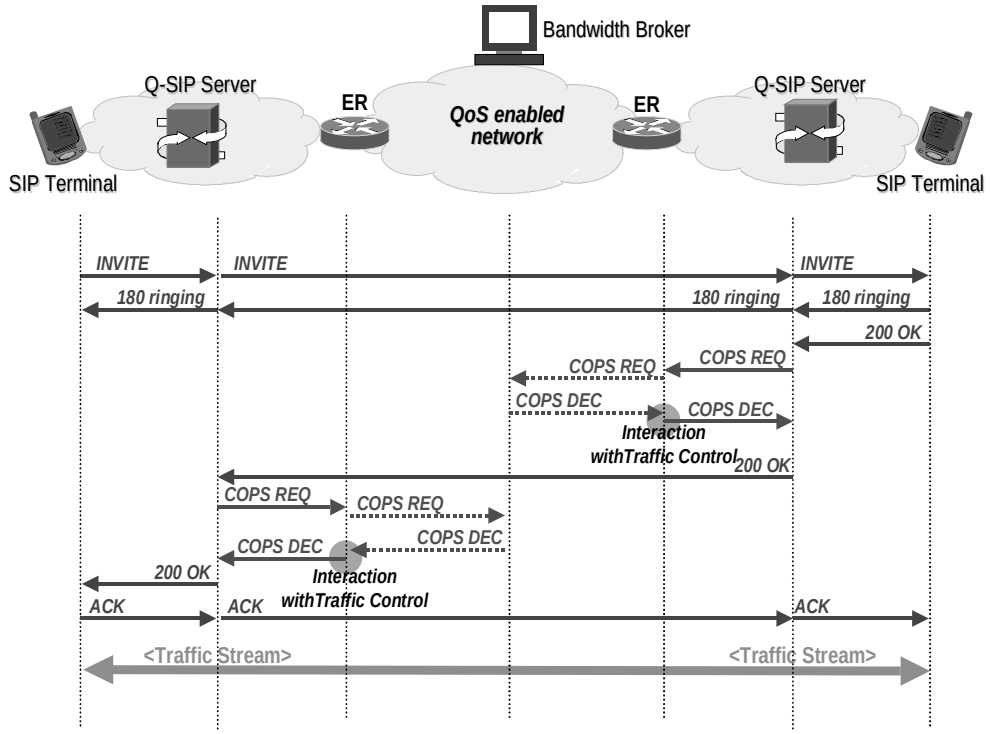

Fig. 4. Q-SIP call signaling flow - QoS enabled model

5. Test-bed implementation

The proposed architecture has been implemented in a test-bed developed in the Nebula project [16]. Some of the DiffServ components of the test-bed have been also discussed in [12].The test-bed, has been shown "up and running" during the GTTI [17] annual meeting in Trieste in June 2002. The overall picture of the test-bed is described in Fig. 5.

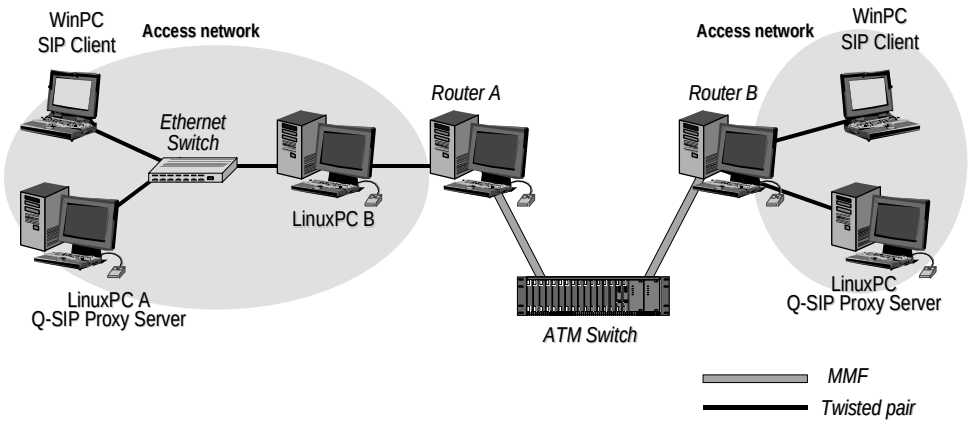

Fig. 5. Overall Nebula test-bed

The Q-SIP proxy servers have been implemented on a Linux PCs based on RedHat7.1 distribution. The Q-SIP server is developed in Java (running on Sun JDK 1.2.2 virtual machine) while the COPS DRA and TC clients/servers are developed in plain C. The TC modules are based on the TCAPI [15] a library that allows the dynamic configuration of filters and scheduling mechanisms. The internal architecture of the test bed elements is shown in Fig. 6. The source code of the Q-SIP server is available under the GPL license [10]. Note that also the COPS-DRA and TC server source code are available under the GPL license. The publicly available "Ubiquity SIP User Agent" version 2.0.10 [13] has been used as SIP terminals, running on Win98 PCs.

The Q-SIP server has a modular architecture, in order to be able to handle different QoS mechanisms. As shown in Fig. 6, there is a JAVA module called Generic Q-SIP Protocol Handler, which is independent of the underlying QoS mechanism. This module dialogues through a JAVA interface with a QoS-specific Interface module (realized in JAVA as well), which is specific of the underlying QoS model. In the picture, the COPS-DRA specific module is shown, which interacts through a socket interface to the COPS-DRA client process, realized in C. The Edge Routers, that act as QoS Access Points, include a COPS DRA server that communicate through a socket interface with a process implementing the Local Decision Server and the COPS DRA client. This process communicates through another socket interface to the TC server that is able to configure the traffic control mechanisms provided by the Linux kernel. Communications between TC server and Linux kernel are made through a netlink socket. The PDP/BB is composed by a COPS DRA server and a Decision Server, that interact through a socket based interface.

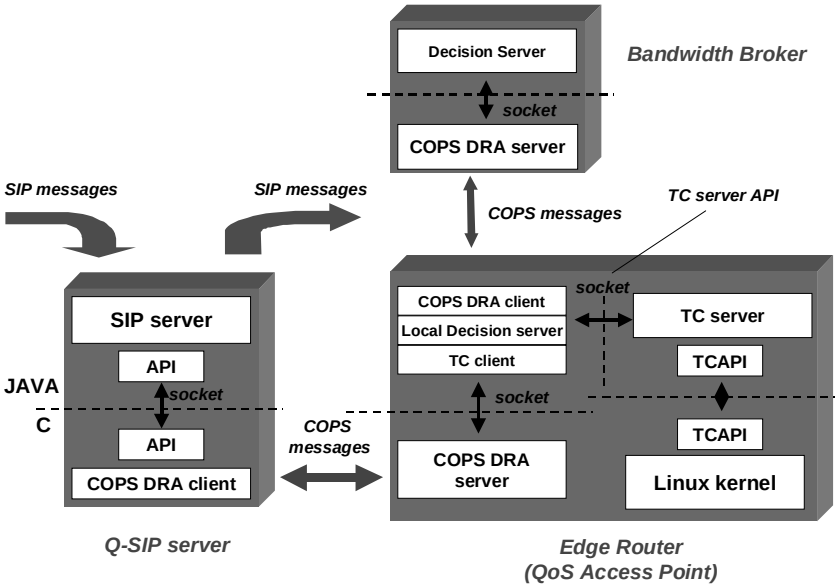

Fig. 6. Q-SIP server, ER, and BB internal architectures

Some tests have been performed on this test-bed to verify the speech quality received at application level in both "*no-QoS*" and "*QoS enabled*" scenarios. In the "*no-QoS*" test we have observed that the quality perceived is good in condition of low traffic but degrades quickly when the background traffic causes the link congestion. In the "*QoS enabled*" test even if an heavy background traffic is present the quality perceived is good and not affected by the traffic volume itself.

The access network A is constituted by a Win98 PC that plays the role of the caller SIP Client and by two Linux PCs. Linux PC A acts as the background traffic generator in both scenarios and, in the "*QoS enabled*" one as the Q-SIP server too. Linux PC B is a router that connects the access network to the backbone link in both scenarios and, in the "*QoS enabled*" one plays the role of the COPS client too. The access network B is constituted by two computers, a Win98 PC and a Linux one directly connected to a router on the backbone link. The Win98 PC is the SIP called client, while the Linux PC acts as the background traffic receiver and, as it concerns the "*QoS enabled*" scenario as both Q-SIP server and COPS client. The router A and B are two Linux PCs on which we have implemented the TC mechanisms. In order to reduce the topology complexity we have decided to make the Router A playing the role of the BB too. The backbone link is emulated by means of an ATM connection using a New Bridge CS-1000 ATM Switch. The next step would be to replace it with the real Internet and, in this case the enhancement of the protocol inter-working should be

considered in order to make the BB configure the core routers when the resource allocation request is accepted.

Further studies have been planned to estimate the packet's end-to-end delay and jitter when the QoS mechanisms have been set-up. Moreover a tool for an objective evaluation of the voice speech quality perceived at user level is an ongoing work.

6. Performance test of Traffic Control API

Performance tests were carried out in order to give figures both on the rate of calls per seconds and on the number of active calls supported by our system architecture.

As described in the previous sections we are using an API to manage the Traffic Control kernel data structures. This means our tests were performed looking at the kernel as a "black-box" while applying to it solicitations. Our aim was to understand the performance limits and not to improve the kernel implementation (though this issue will be briefly addressed in the following). We are aware that the results presented here may not have a general validity since they rely on the PC hardware on which the software implementation is running. However, it is important to notice that while the absolute values presented are likely to be variable depending on the processor speed and memory size, the relative values are likely to be valid across multiple speed processors and memory sizes. The results presented here are relative to an IA32 PC (Pentium IV) with 256 MB of RAM; moreover the kernel used was the 2.4.19.

The first set of tests performed were focused to compute the resource allocation request time as it is seen from the Traffic Control server point of view. To this aim we profiled the Traffic Control server code extracting the time it takes for a resource allocation request to be completed (in this test all the resource allocation requests we issued were accepted). Fig. 7 reports the time needed for an allocation request to be completed (different rates of resource allocation requests are plotted) versus the number of requests already allocated in the systems. From the graph, it's easy to notice that this time is independent on the rate of the calls (in this test every new call makes a new resource allocation request start) while it depends on the number of requests already allocated only. Moreover, the graph shows how the resource allocation time grows with the number of requests already accepted by the system. This result may indicate a linear management of the filter/class inside the kernel through a

chain of data structures (to achieve higher performance this linear management has to be avoided; an hash tabled should be used instead).

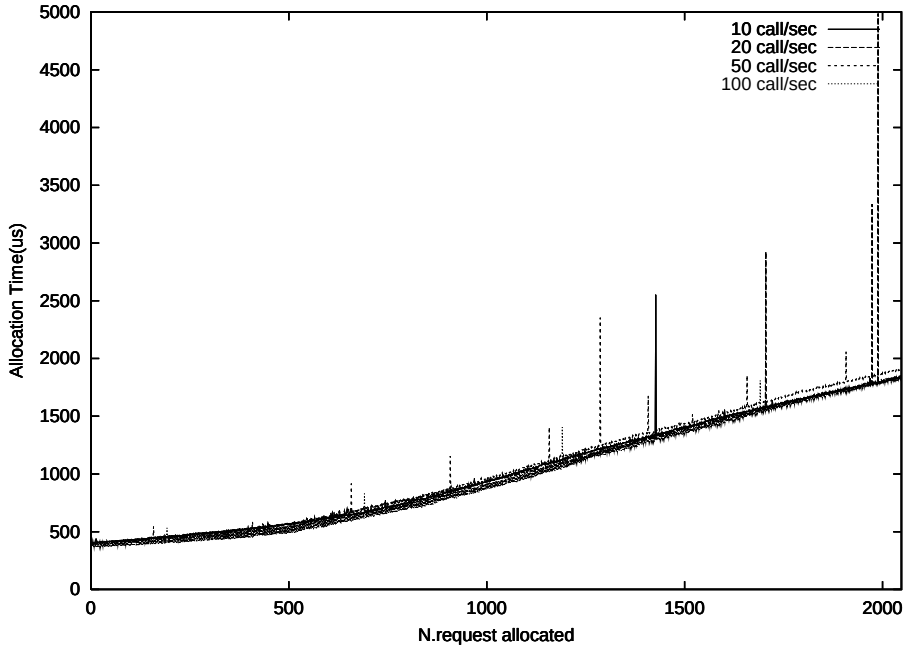

Fig. 7. Resource Allocation Request time versus resource allocation request number

When performing the previous tests we realized that, if the allocation time depends only on the state of the system when the allocation request takes place (i.e. on the number of requests already allocated in the system so far), there is a bound to the number of simultaneous calls that can be handled at different allocation rates. This limit is exceeded when the inter-arrival allocation request time is greater than the time required by the Traffic Control server to serve a resource allocation request. We report those results in Table 1 as performance bound since this is the value that, if exceeded, makes the resource allocation server starting buffering resource allocation requests.

The performance bounds gets the more tightening as the request allocation rate increases. The results reported here have to be interpreted as the maximum number of simultaneous active calls our current implementation may handle with a given call rate.

Calls/sec (allocation requests/sec)	Maximum number of simultaneous allocated requests
1	>10000
2	6246
5	4419
10	3743
20	2914
50	2048
100	2048
200	2048
500	2048
1000	992

Table 1. Maximum number of simultaneous allocated requests versus calls per second

This number of simultaneous calls decreases as the call rate increases up to the maximum supported call rate of approx 2000 call/sec (as it can be seen from Fig. 7, the time for the first resource allocation request is in the range of 0.5 msec independently on the call rate). Moreover, Table 1 reports a strange behavior when the number of requests already allocated is 2048: for resource allocation request rates ranging from 50 to 500 calls per second the performance limit is always 2048 allocated requests. This strange behavior is highlighted in Fig. 8 where we report a slice of our test with call rate of 100 calls/sec with request number ranging from 1000 to 3000.

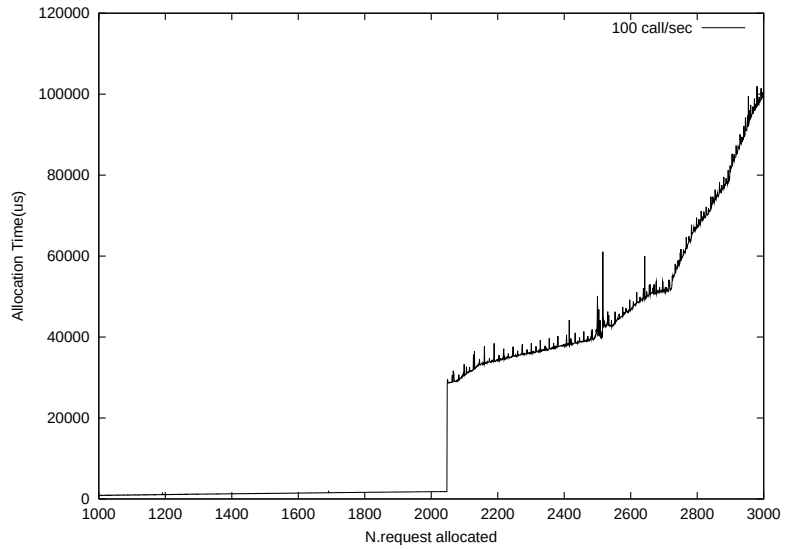

Fig. 8. Resource Allocation Request time versus resource allocation request number

The resource allocation time is still increasing but Fig. 8 shows that the performance worsens of one order of magnitude (from approx 2 msec to more than 20 msec) when exceeding the 2048th request allocated. This behavior is common to all the call rate tested ranging from 1 to 1000 call/sec. This result may indicate that table entries are allocated from a dedicated kernel memory pool up to a certain number; when this number is exceeded then the kernel needs to allocate extra memory space addressing a different memory area thus loosing efficiency (a higher amount of memory space is required to be reserved in kernel space directly to manage larger systems). The last tests performed were devoted to understand whether such a performance worsening is reversible once the number of allocated requests drops below 2048. In Fig. 9 and Fig. 10 we report two different tests:

- in the former (Fig. 9) we firstly allocated 4000 requests, then we released them and then we allocated again 4000 requests (here the requests with numbers [0-4000] and [8000+] are resource allocation requests, the others are resource release requests);
- in the latter (Fig. 10) we firstly allocated 400 requests, then we released them and then we allocated again more than 2000 requests (here the requests with numbers [0-400] and [800+] are resource allocation requests, the others are resource release requests);

In Fig. 9 is clear that the performance worsening affects the resource allocation requests time only (releasing a resource previously allocated is always done within a little time) and that, once the limit of 2048 requests allocated is reached, the system has no reversibility even if the number of resource allocated decreases to 0 (the 8001st request is a resource allocation request and the time it takes to complete is the same as the 2049th in this test). In Fig. 10, instead, the performance degradation took place only when the request allocated exceeded 2048 for the first time: i.e. when the number of the request was 2848 (= 400+400+2048).

Taking into account the results shown here, we can claim that, after testing our system under the load of different resource allocation rates:

- no dependency on the call rate value was found;
- the number of simultaneous calls supported by the system depends on the call rate following an inverse law, except for the specific behavior, above discussed, due to kernel memory management;
- to achieve better performance the number of 2048 simultaneous requests allocated must not to be exceeded.

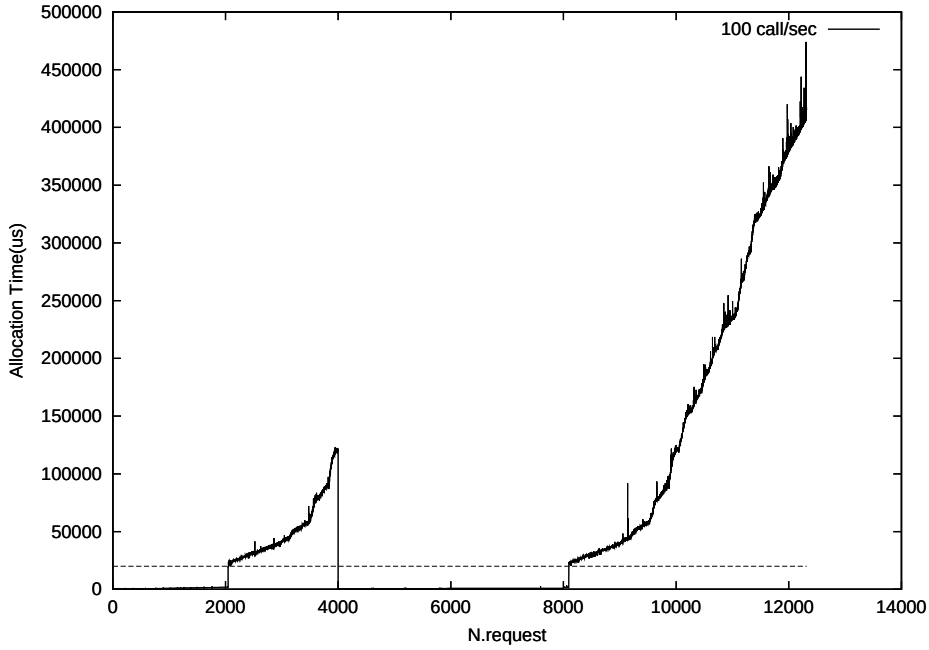

Fig. 9. Resource Allocation/Release Request time versus request number (exceeding the limit)

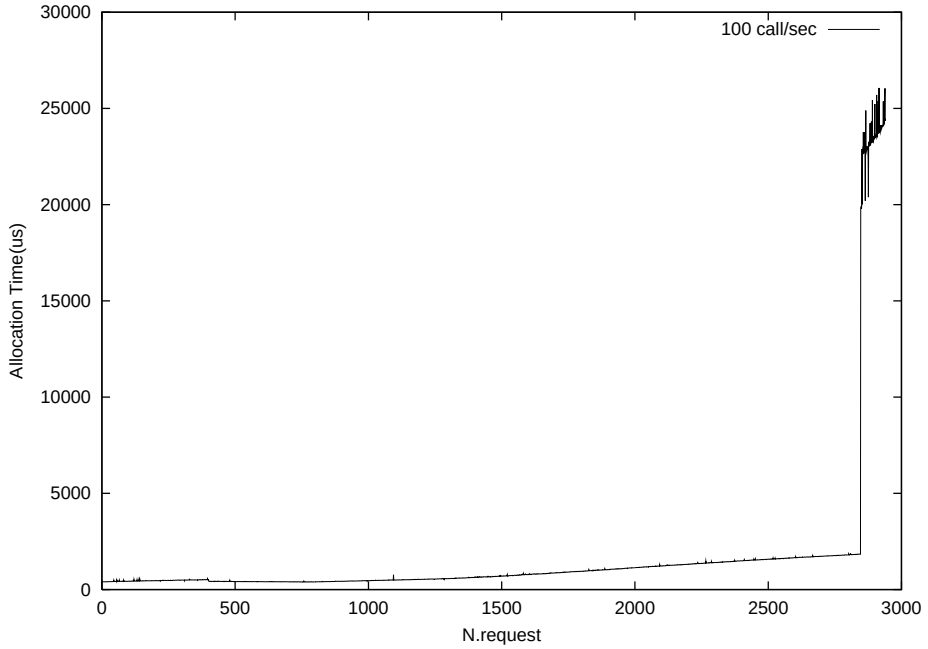

Fig. 10. Resource Allocation/Release Request time versus request number (not exceeding the limit)

7. Conclusions

In this paper we have proposed an architecture for the dynamic configuration of DiffServ QoS mechanisms in IP network. We started from the definition of a “TC server API” to interact with QoS mechanism in an open source router based on Linux OS. Then we have considered the COPS-DRA protocol as a generic signaling mechanism to let QoS client interact with a QoS enabled IP network. The interaction between the COPS-DRA control logic and the TC server API has been defined. As an example application, we have considered the IP telephony based on SIP protocol. The extension to SIP protocol required to interact with the COPS-DRA mechanisms have been defined. We introduce modification in SIP protocol that only affects SIP “proxy” servers, so that “legacy” SIP user application can be fully reused. We note also that the solution is fully backward compatible with current SIP based equipment that does not support QoS, allowing a smooth migration. The whole architecture has been implemented in a testbed and the internal architecture of the software modules is described. Results on the performance of Traffic Control mechanisms in Linux based routers have been reported. Future work include the combined performance evaluation of signaling mechanisms and traffic control mechanisms in the router, in order to understand the scalability in terms of call rate and number of active calls also including the signaling load.

Acknowledgements

The authors would like to thank Donald Papalilo and Enzo Sangregorio for their work in support of the specifications and of the test-bed implementation.

References

[1] X. Xiao, L.M. Ni “Internet QoS: A Big Picture”, IEEE Networks, March 1999

[2] W. Zhao, D. Olshefski and H. Schulzrinne “Internet Quality of Service: an Overview” Columbia University, New York, New York, Technical Report CUCS-003-00, Feb. 2000

[3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler “ SIP: Session Initiation Protocol”, IETF RFC 3261, June 2002

[4] M. Garcia-Martin “3rd-Generation Partnership Project (3GPP) Release 5 requirements on the Session Initiation Protocol (SIP)”, <draft-ietf-sipping-3gpp-r5-requirements-00.txt>, October 2002, Work in Progress, http://www.ietf.org/internet-drafts/draft-ietf-sipping-3gpp-r5-requirements-00.txt

[5] G. Camarillo et al. “Integration of Resource Management and SIP”, IETF RFC 3312, October 2002.

[6] K. Chan et al. “COPS usage for Policy Provisioning (COPS-PR)”, IETF RFC 3084, March 2001.

[7] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry “The COPS (Common Open Policy Service) Protocol”, IETF RFC 2748, January 2000

[8] S. Salsano "COPS usage for DiffServ Resource Allocation (COPS-DRA)", <draft-salsano-cops-dra-00.txt>, September 2001, Work in Progress, http://www.coritel.it/projects/cops-bb

[9] S. Salsano, L. Veltri “QoS Control by means of COPS to support SIP based applications”, IEEE Network, March/April 2002

[10] L. Veltri, S. Salsano, D. Papalilo, “QoS Support for SIP based Applications in DiffServ Networks”, <draft-veltri-sip-qsip-01.txt>, October 2002, Work in Progress, http://www.coritel.it/projects/qsip

[11] L. Veltri, S. Salsano, D. Papalilo, “QoS Support for SIP Based Applications in a Diffserv Network”, IEEE Softcom 2003, October 7-10, 2003, Split, Dubrovnik (Croatia), Ancona, Venice (Italy).

[12] W. Almesberger, S. Giordano, R. Mameli, S. Salsano, F. Salvatore “A prototype implementation for IntServ operation over DiffServ Networks”, IEEE Globecom 2000, S. Francisco, December 2000

[13] “SIP User Agent”, Ubiquity Software Corporation, http://www.ubiquity.net

[14] B. Hubert “Linux Advanced Routing & Traffic Control HOWTO” http://lartc.org/howto

[15] TC API Project - http://www-124.ibm.com/developerworks/projects/tcapi

[16] Nebula Project Home Page - http://nebula.deis.unibo.it/

[17] GTTI Home Page - http://www.gtti.cnit.it/

[18] Levi, Meyer, Stewart "SNMP Applications", IETF RFC 3413, December 2002.