Supporting QoS for Legacy Applications

C. Tsetsekas, S. Maniatis and I. S. Venieris

National Technical University of Athens,
Department of Electrical and Computer Engineering,
9 Heroon Polytechniou str, 15773,
Athens, Greece

{htset, sotos}@telecom.ntua.gr,
ivenieri@cc.ece.ntua.gr

Abstract. Internet is widely known for lacking any kind of mechanism for the provisioning of Quality of Service (QoS) guarantees. The Internet community currently concentrates its efforts on mechanisms that support QoS in various layers of the OSI model. Apart from that, the Internet community is trying also to define the protocols, through which applications and users will signal their QoS requirements to the lower network layer mechanisms. The latter task, however, is not trivial, especially for legacy applications that cannot be modified and recompiled. This paper presents a framework for a middleware component that supports QoS for legacy applications. It mainly focuses on the support of a proxy-based framework for the identification of flows, the measurement of basic QoS parameters and the definition of an API that can be used by middleware components or even applications. The position of this proxy architecture in a reference network topology and the communication with other middleware entities is also discussed.

1 Introduction

The Internet has had an overwhelming effect on the way people interact and communicate. The Internet is based on the Internet Protocol (IP) that provides a simple, easily deployable and best effort in nature network service. The tremendous growth of IP has also boosted the development of various IP-based applications that include complex, quality-intensive multimedia services. As a result, the Internet community has been heavily engaged with defining the appropriate mechanisms that will provide Quality of Service (QoS) support for applications. It is foreseen that a proper combination of these mechanisms will eventually provide ubiquitous end-to-end QoS.

To be more specific, the Integrated Services (IntServ) [1] and the Differentiated Services (DiffServ) [2] are two of the mechanisms proposed by IETF. In IntServ, network resources, which are mainly defined in terms of bit rate, packet delay and maximum transfer size, are reserved in every node along the path from the sender to the receiver. In contrast to the IntServ model, which uses explicit resource reservation for every flow requesting QoS, thus raising scalability concerns, the DiffServ model is

a simpler and straightforward architecture, which relies on prioritization of some flows over others. However, the simplicity of the DiffServ model and its lack of mechanisms for the systematic and automated resource allocation necessitated the introduction of the Bandwidth Broker concept [3]. The Bandwidth Broker is a logical entity, complementing the DiffServ infrastructure, which is responsible for performing policy-based admission control, managing network resources, and configuring specific network nodes, among others.

Except for the aforementioned network mechanisms, applications and users have to be able to indicate their QoS requirements to the network entities through the proper interface mechanisms. Currently, the Resource ReSerVation Protocol (RSVP) [4] provides the interface to setup and control QoS in the IntServ model. There are a lot of efforts to utilize the RSVP protocol along with other QoS technologies like DiffServ. Such protocols may be invoked with the use of APIs, like the Generic QoS API integrated in WinSock2 from Microsoft, and the QoS Application Programming Interface (API) from the Internet2 community [5]. The existence of these APIs presupposes that the applications must be modified, and compiled again in order to take advantage of them.

The main motivation behind this paper is the support of legacy applications. The individual characteristic of such applications is primarily that they cannot be modified. So they cannot directly make use of one of the aforementioned APIs. Moreover, the IP port numbers are usually not known a priori, because they are negotiated dynamically. In addition, any modification of the network QoS provisions during the lifetime of the application cannot be communicated to the application, so that it cannot react to them. In order to alleviate these inherent limitations of legacy applications, we propose to make use of a middleware component that acts as an agent between the applications and the network QoS entities. This paper presents the framework of the middleware for the support of any kind of application over a QoS-enabled IP network. The main responsibility of the middleware is to provide the mechanisms for the description and selection of QoS parameters and the forwarding of QoS requests to the appropriate network entities. Moreover, the paper describes to a great extent a proxy framework for the support of fundamental operations, like the detection of new flows and the measurement of their traffic profile, as well as additional features, like the transparent support of RSVP, and the identification of various multimedia streams (video, audio) within a Web session.

The paper is structured as follows. Section 2 gives a brief overview of the overall context within which this work is being accomplished. It presents the general Aquila [6] concept and, more specifically, the End-User Application Toolkit. Section 3 addresses the proxy framework, identifying the problems and proposing solutions. Finally, section 4 presents the conclusions.

2 The AQUILA Architecture

The Aquila project [6] aims to define, implement and evaluate an enhanced architecture for dynamic end-to-end Quality of Service support over IP networks. Existing

approaches to QoS specified for the Internet, such as IntServ, DiffServ and MPLS are used as a basis, and the solutions implemented are verified and tested within trials involving end-users.

The Aquila network architecture (Figure 1) is created by the interconnection of various administrative domains controlled by different Internet Service Providers (ISPs). These domains are distinguished into two categories: core and access networks. In the core network, IP flows receive prioritized treatment over others with the adoption of the DiffServ architecture. The access network connects hosts to the core Internet, through Edge Routers that perform enhanced functionality compared to usual core routers. To be more specific, in the Aquila architecture, in the Edge Router of the ISP where an access network is connected, user packets are classified, shaped and policed. The Edge Router also marks packets and aggregates their corresponding flows into groups under the same DiffServ Codepoint.

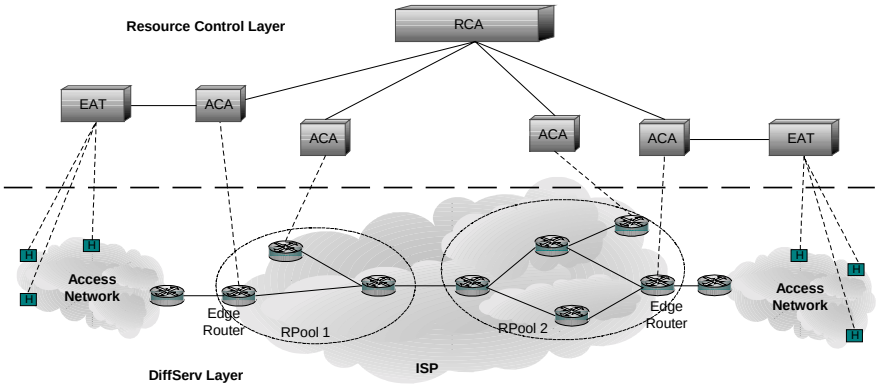

Figure 1: Aquila Network Architecture

The main innovation of the Aquila architecture is a new layer on top of the DiffServ core ISP network, called the Resource Control Layer (RCL) [7]. The RCL is responsible for the management of network resources. It resembles a distributed Bandwidth Broker in the DiffServ architecture. However, this architecture extends the Bandwidth Broker model in two aspects: it caters for scalability by logically dividing the administrative domain in sub-areas, called Resource Pools (RPool), and it provides dynamic end-to-end support for QoS.

The Resource Pools construct a tree hierarchy of RPools. The root of this tree is the whole administrative domain, while the leaves are the edge routers. The Resource Control Agent (RCA) is the central entity of the RCL, responsible for the overall control of the administrative domain. The RCA operates on a long-term basis, by managing and distributing the resources of the domain to the Admission Control Agents (ACA), by exploiting the Resource Pool concept. An ACA is associated with each edge router of the domain, and it is always a Resource Pool Leaf. It operates on request basis, by performing policy and admission control and granting a share of resources to individual flows in response to a QoS request. In order to perform admission control, it intelligently compares the requested resources with the total amount of

the available resources granted to it. The resource management is performed with the aid of an intelligent algorithm that caters for the initial distribution and re-distribution of resources during the course of the network operation [8].

The End-user Application Toolkit (EAT) [9] provides end-to-end QoS support. Applications and end users can make their own QoS requests with the use of the EAT. The major objective of the EAT is to provide a scalable and efficient approach for transparent QoS support for multimedia applications. Existing commercial applications leverage the EAT functionality without the need of any modifications. Resource reservation requests may be sent automatically, without the need of user intervention. Therefore, the end-users are shielded from the application complexity, as well as from QoS related aspects, such as traffic specification.

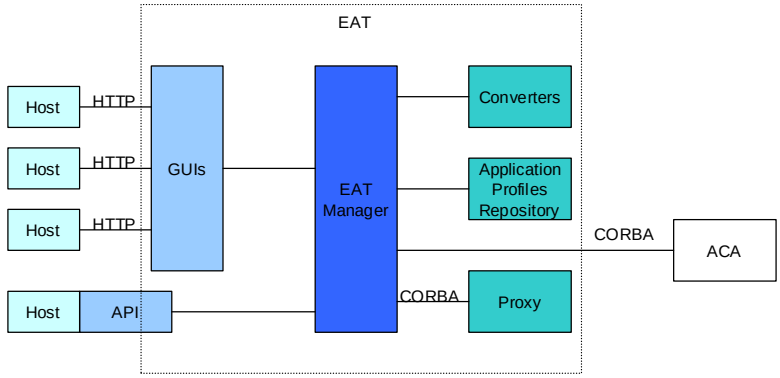

Figure 2: The End-User Application Toolkit

The main building blocks of the EAT (Figure 2) are the EAT Manager, the Proxy module, the Converter modules, the Application Profiles Repository, the QoS Application Programming Interface (API) and the Graphical User Interfaces (GUIs). The EAT Manager coordinates all other modules, to ensure the smooth operation of the toolkit. Moreover, it is responsible for forwarding the reservation requests to the ACA. For this purpose it utilizes an interface based on CORBA [10].

The Converter modules intelligently and automatically prepare reservations for selected application sessions (e.g. a video session), with the appropriate QoS parameters. In order to be able to perform such an operation, the Converters rely on the concept of *Application Profiles*. An application profile describes the QoS requirements of a specific application based on a Data Type Definition (DTD) scheme. The concept of application profiles stems from the fact that each individual application can be thoroughly tested in order to discover its QoS requirements. After the testing of the application, the application profile is composed by the measured parameters, using the eXtended Markup Language (XML) format [11], and checked upon the defined DTD. Application Profiles are stored in a repository, which is queried by the Converters upon a QoS request.

The GUIs give the opportunity to the user to place and release reservations, as well as to monitor the current QoS conditions, through a common web browser. The API provides a library of functions that can be used by developers to provide QoS to new

applications. Finally, the Proxy module addresses the inherent problems of legacy applications described in the introduction. The Proxy module is based on a well-defined framework that is briefly discussed in the rest of the paper.

The EAT modules have clear and consistent interfaces among them. This enables one module to be used outside of the context of the Aquila architecture.

3 The Proxy Framework

The Proxy module has been designed and built under the consideration of being independent of specific architectures and operating systems. It provides an Application Programming Interface (API) via CORBA [10] to allow any application or middleware to take advantage of its functionality. Therefore, it is not adequate only for the Aquila architecture, but for any architecture that needs support for legacy applications.

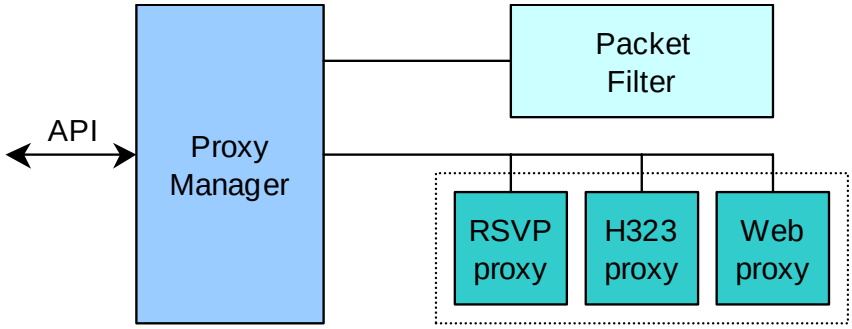

Figure 3: The Proxy Framework

The Proxy Framework is graphically depicted in Figure 3. The Proxy Manager is the central entity that takes control of the overall operations. It co-ordinates all the components of the Proxy Framework by identifying the available Proxies and by making them available to the applications. It also keeps track of all discovered QoS flows through a *Flow Database*. The entries of this database are composed of IP flows identified from the Proxies along with measurement information.

However, the most important components of the Proxy Framework are the Application-Level Proxies and the Packet Filter. They are described in detail in the following paragraphs.

Application-Level Proxies

The main functionality of the Proxy includes the identification of flows that may need QoS treatment and the measurement of their basic QoS parameters. For this purpose, the Proxy Framework includes Application-Level Proxies that perform signaling protocol translation in an effort to extract the parameters of a flow that are important for a

reservation request (addresses and port numbers). A specific Application-Level Proxy has been specified for each major protocol that is used for connection set up (SIP, H.323, RTSP etc.). The implementation of such a Proxy is based on a generic framework that specifies the interfaces it should implement and offers already established components that make development easier.

The H.323 application proxy can serve here as an example: Currently, there are many applications that make use of the H.323 protocols [12], like NetMeeting in Windows platforms. The need for such a proxy stems from the fact that the ports used for the audio and video streams are not standard, but ephemeral. The H.323 protocol makes use of the Q.931 and H.245 signaling protocols (over TCP) to set up the RTP and RTCP audio and video flows. The task of the Proxy in this case would be to intercept the exchanged H.245 control messages, in order to find out the dynamically negotiated ports used by these streams. The Session Initiation Protocol (SIP) [13] Proxy also operates in the same way. The SIP messages exchanged through the SIP Proxy who, process them and extracts information about the addresses and ports of the audio connection. The Proxy may also extract information relevant to resource reservation (see the discussion on SIP extensions [14]) and use it for the formulation of a resource request to the Resource Control Layer of Aquila.

A Web Proxy is also used for identification of Web traffic that may need QoS support. Such traffic would be links to multimedia content. The Proxy steps into the transmitted content and identifies possible audio or video streams that could need QoS treatment. Through the API, it communicates the acquired streams to the responsible entity to take care of reservations. In the Aquila context, the Proxy Manager, after being triggered by the Web proxy about specific multimedia streams, contacts the EAT Manager through the API. According to user-specified settings, the EAT Manager can either directly place a reservation with the aid of the Converter modules, or contact the end-user through the GUI to ask for confirmation.

An important component of the Proxy Framework is an RSVP proxy, responsible for identifying and interpreting RSVP protocol messages. In the Aquila architecture, this is very significant, because the core network does not support IntServ for scalability reasons. Therefore, while the PATH and RESV messages of RSVP are transparently forwarded in the Aquila core network, they are caught at the edges by the RSVP Proxy. Their content (Flowspec and Filterspec) is extracted and forwarded to the EAT for the initiation of a new reservation. In this way, legacy RSVP applications can be transparently accommodated by our architecture. It is obvious that any architecture could make use of the RSVP proxy for similar reasons, in order to decouple RSVP from core network QoS technologies.

Based on the Proxy Framework, new Proxies may easily be implemented in order to cover for the needs of other legacy applications. However, the functionality of the Framework is not restricted to identifying flows pertaining to a specific protocol. A central component, the Packet Filter is used to detect new flows that have not already been registered by the Proxies.

Packet Filter

The Packet Filter is situated at the borders between access and core networks, usually inside a firewall. It captures all incoming and outgoing packets from the access network, checking their source and destination address. When a new flow is detected (and no Proxy has been used), the Packet Filter examines its packets, in order to extract important information: IP addresses and port numbers, as well as hints on their content, mainly through RTP headers in the case of multimedia flows. The Packet Filter can therefore provide support for flows that are not established with the use of a supported signaling protocol. Upon the detection of such a flow the EAT is contacted in order to decide whether this flow will receive QoS support or not.

However, the main functionality of the Packet Filter is to conduct periodic traffic measurements. The value of such measurements is two-fold. First, the traffic profile of QoS flow may be estimated. Upon the establishment of a new connection, measurements can provide some preliminary values for the formation of a resource reservation request. As time passes by and the flow of the IP packets reaches a more stable rate, more accurate measurements can be available and corrections to the QoS request may be made. This feature of the Packet Filter enables the *adaptation* of a QoS reservation according to the long-term fluctuations of user traffic.

The second advantage that measurements offer is that users may receive immediate feedback on their traffic usage and on whether the requested levels of QoS reservation are honored by the network. In the Aquila architecture, the flow identifiers and the measurements are passed through the Proxy Manager and the API to the EAT Manager. The EAT then displays the performance of the network in a comprehensible format to the end-user through the GUI.

The basic measurements conducted by the Packet Filter mainly include the estimation of the Tspec of a flow, as it is defined within the RSVP protocol [5]. The Tspec consists of the following parameters: Peak rate, average rate, burst size, minimum policed unit and maximum packet size. Those parameters are estimated by processing a list of packet sizes along with their corresponding timestamp. The peak rate is calculated by dividing the sum of bits sent over a small period of time by this period. The same way is used for the average rate, by using a substantially longer period of time. The burst size is estimated as the excess bytes sent over a small period of time, by subtracting the bytes that should have been sent according to the estimated average rate, from the actual bytes sent over this period. Finally, the minimum policed unit and maximum packet size is the minimum and maximum packet size that has been observed for the specific flow.

A future extension of the measurements functionality will include the estimation of the delay, jitter and packet loss values of a packet flow. Those parameters cannot certainly be measured by a standalone Packet Filter, but require the co-operation of two measurement entities. A sender and a receiver should exchange test packets with characteristics (packet marking, source and destination networks) similar to the flows under measurement. Therefore we can measure the delay and its variation. However, this solution poses new open questions, the most important being how the two Proxies will locate each other and communicate.

The Proxy API

The Proxy Manager offers an API to the EAT using CORBA. In the Aquila architecture, the EAT Manager uses this API to receive flow information and measurements. Moreover, the API provides upcalls, that enable it to signal to the EAT a significant change in the Tspec that may require the adaptation of the reservation. This approach was chosen in order not to burden the system's operation with a flood of signaling messages between the EAT Manager and the Proxy.

This API may also be used by other external entities, such as applications. Through the API, an application can query the Packet Filter about flow identifiers and measurements. Moreover, it can query, tune, or configure existing proxies, or even add new proxies to the framework to cater for individual needs.

Performance Issues

The most obvious location for the Proxy Framework is inside a firewall. In this way, it can leverage the functionality of packet filtering in the edge routers. The operation of a packet filter at the ingress of the core network is not expected to create a bottleneck. We believe that the deterioration of performance of the network will not be greater that the one introduced by a simple firewall. However, if the performance deterioration during the operation of the Proxy is significant, more than one Proxy can be introduced in the network, by serving separate sections of an administrative domain.

On the other hand, one of the advantages of the Proxy Framework concept is the provision of QoS to flows in the case of firewalls and Network Address Translation (NAT). Since the Proxy Framework is exactly in the place where address translation operations take place, it can identify the parameters of a flow (addresses and port numbers) from both sides of a firewall.

4 Conclusions

In this paper we have presented a Proxy Framework for the support of legacy applications within a QoS architecture. The Proxy Framework is an important feature of Aquila, an architecture that enables dynamic end-to-end QoS support over a DiffServ enabled core network.

The basic functionality of the Proxy includes the use of Application-Level Proxies that perform protocol translation of major connection set up protocols, in an effort to support QoS for legacy applications. Moreover, the Proxy Framework is enhanced with packet filtering capabilities that enable the performance measurements as well as estimation of the traffic profile of QoS flows.

Acknowledgement

This work was performed in the framework of IST Project AQUILA (Adaptive Resource Control of QoS Using an IP-based Layered Architecture - IST-1999-10077) funded in part by the EU. The authors wish to express their gratitude to the other members of the AQUILA Consortium for valuable discussions.

References

1. R. Braden et al., “Integrated Services in the Internet Architecture: an Overview”, RFC 1633
2. S. Blake et al., “An Architecture for Differentiated Services”, RFC 2475
3. K. Nichols et al., “A Two-bit Differentiated Services Architecture for the Internet”, RFC 2638
4. R. Braden et al., “Resource ReSerVation Protocol (RSVP)”, RFC 2205
5. B. Riddle & A. Adamson, “A Quality of Service API Proposal”, http://apps.internet2.edu/qosapi.htm
6. The Aquila project: http://www-st.inf.tu-dresden.de/aquila/
7. G. Poliths, P. Sampatakos, I.S. Venieris, “Design of a multi-layer bandwidth broker architecture”, Lecture Notes in Computer Science; Vol 1938, Springer Verlag, Oct 2000
8. E. Nikolouzou, P. Sampatakos, I.S. Venieris, “Evaluation of an Algorithm for Dynamic Resource Distribution in a Differentiated Services Network”, Proceedings of ICN2001, June 2001
9. Ch. Tsetsekas, S. Maniatis, I.S. Venieris, “An end-to-end middleware solution for the support of QoS in the Internet”, Proceedings of SoftCOM2000, Oct 2000
10. CORBA/IIOP 2.3.1 Specification, http://www.omg.org/ corba/cichpter.html
11. E.R. Harold: XML Extensible Markup Language, IDG Books Worldwide, 1998
12. ITU-T Recommendation H.323, Packet-Based Multimedia Communication Systems, 1998
13. M. Handley et al., “SIP: Session Initiation Protocol”, RFC 2543
14. W. Marshall et al., “Integration of Resource Management and SIP”, draft-ietf-sip-manyfolks-resource-00, November 2000