A Middleware Solution for the Support of QoS for Legacy Applications

C. A. Tsetsekas, S. I. Maniatis, I. S. Venieris
National Technical University of Athens,
Department of Electrical and Computer Engineering,
9 Heroon Polytechniou str, 157 73, Athens, Greece
Telephone: (+30 - 1) 77 22 551, FAX: (+30 - 1) 77 22 534
E-mail: {htset, sotos}@telecom.ntua.gr, ivenieri@cc.ece.ntua.gr

Abstract - The evolution of quality of service mechanisms in the Internet is prominent nowadays. The Integrated and Differentiated services, along with the evolving Bandwidth Broker concept complement each other to offer the required quality. One significant aspect towards this direction is the capability for users and applications to convey their QoS requests to the core network entities. This paper discusses a middleware architecture that seamlessly supports legacy non-QoS-aware applications to take advantage of the quality provisions. The proposed middleware can also leverage existing QoS-aware or newborn applications. It is destined to cooperate with the QoS management entities of the Aquila architecture in the access and core networks, even though it can also be used in conjunction with existing architectures.

Keywords – QoS Middleware, DiffServ, Bandwidth Brokers, Legacy Applications, Packet Filter

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 vary from simple FTP or electronic mail programs to complex, quality-intensive multimedia ones.

The Internet Integrated Services (IntServ) [1] were defined by the IETF in an effort to bring Quality of Service (QoS) into the IP world. In IntServ, network resources, which are 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 raising scalability concerns, the Differentiated Services (DiffServ) model [2] 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 [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. Moreover, the introduction of the Bandwidth Broker concept made evident the need for a ubiquitous procedure to request QoS parameters. Standard interfaces should be defined between applications and the resource management entities. New applications should be developed that will use those interfaces, in order to communicate their QoS requests.

Furthermore, there is a vast amount of legacy commercial applications that do not have any inherent QoS support. This issue is addressed by the concept presented in this paper, which discusses a middleware that provides Quality of Service support for legacy applications over a QoS-enabled network.

The paper first discusses in brief the overall proposed network architecture in section 2. Section 3 discusses in detail the middleware component at the access network in terms of requirements, functionality, architecture, and interaction with other components. A few implementation issues are discussed in section 4. Last, in section 5 the conclusions are presented along with future work.

2. NETWORK ARCHITECTURE

The Aquila project [4] 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 [5] will be used as a basis, and

the solutions implemented will be verified and tested within trials involving end-users.

The Aquila network architecture 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 Differentiated Services (DiffServ) architecture. In the access network, which connects hosts to the Internet, user packets are classified, shaped and policed by the Edge Router, which is the point where an access network is connected to its Internet Service Provider (ISP). The Edge Router also marks packets and aggregates their corresponding flows into groups under the same Codepoint. A DiffServ Codepoint is a value which is encoded in the DiffServ field (the six most significant bits of the IPv4 ToS octet or the IPv6 Traffic Class octet) of a packet, and which each DiffServ capable node must use to select the appropriate treatment for this packet.

The main innovation of the Aquila architecture is a new layer on top of the DiffServ network, called Resource Control Layer (RCL). The RCL is a QoS middleware that is responsible for the management and allocation of network resources to individual flows. This layer can be also viewed as a distributed Bandwidth Broker in the DiffServ architecture. However, this architecture extends the Bandwidth Broker model in two aspects: it provides dynamic end-to-end support for QoS, while including the end-hosts in the middleware.

The **Resource Control Agent (RCA)** is the central entity of the RCL, responsible for the overall control of an 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)**. The domains of the core network are divided in areas, the routers of which are controlled by an ACA. The ACAs operate on request basis by performing policy and admission control and granting a share of resources to individual flows in response to a QoS request.

End-users can make their own QoS requests with the use of the **End-user Application Toolkit (EAToolkit)**. The EAToolkit is a distributed framework residing in the access network that enables non-QoS aware applications to make resource reservation requests to the RCL. The EAToolkit, which is the focus of our work, is presented in extent in the rest of the paper.

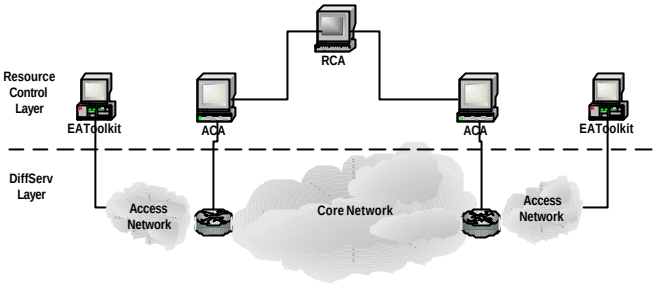

Figure 1: Network Architecture

3. THE END-USER APPLICATION TOOLKIT

The EAToolkit architecture provides a framework for the support of Quality of Service for multimedia applications in the Aquila architecture. Its responsibility is to provide the mechanisms for the description and selection of Quality of Service parameters and the forwarding of reservation requests to the Resource Control Layer. The EAToolkit performs the role of a QoS Middleware, which hides low-level complexity and heterogeneity and presents users and applications with simple and uniform procedures for the reservation of resources in the network.

The motivation for the conception of the EAToolkit was the need to support Quality of Service for legacy multimedia applications (streaming application, video conferencing and Voice over IP applications). With the term "legacy", we refer to existing commercial applications that are not QoS-aware, but also to applications that can already make resource requests to the network with the use of well-known mechanisms, such as the Resource Reservation Protocol (RSVP) [6]. Those applications are usually very complex in opening network connections and exchanging control and user data. Moreover, their source code is not available, so that they cannot be modified to make use of a QoS request mechanism. Our target is to provide the means to those applications to use the services of the Aquila Resource Control Layer (RCL) in a transparent way, without modifying them. No modifications should also be necessary for operating systems.

While the chief purpose of the EAToolkit is the support of QoS for legacy multimedia applications, new applications can also benefit from the Aquila architecture. The EAToolkit supplies an Application Programming Interface (API) that enables new applications to directly make QoS requests to the network.

3.1 Objectives-Functionality

The major objective of the EAToolkit is to provide a scalable and efficient architecture for transparent QoS

support for multimedia applications. Existing commercial applications will leverage the EAToolkit functionality without the need of any modifications. Resource requests may be sent automatically, without the need of user intervention. Therefore, the end-user is shielded from the application complexity and it is not important for him/her to be aware of QoS related aspects, such as traffic specification.

Nearly all proposed QoS Middleware architectures presuppose the creation of new applications, in order for them to be able to use the functionality offered. These applications use proprietary Application Programming Interfaces (APIs) for the invocation of QoS support in the Middleware. On the contrary, we propose a solution ready to be used with existing applications. This architecture does not have as a prerequisite the development of tailor-made applications that comply to its individual characteristics.

It should be noted also, that although the EAToolkit is designed to leverage the Aquila architecture functionality, it might also be used in conjunction with other QoS architectures, such as IntServ with RSVP as well as the Bandwidth Broker.

In order to make a QoS reservation to the Aquila Resource Control Layer, or to other QoS architectures, we mainly need the following information:

- Local and remote addresses and ports of the connection(s)
- Traffic specification of the IP flow
- QoS requested
- Reservation Time

The local and remote addresses and ports of the multimedia connections should be provided to the RCL, since core routers depend on them to provide prioritized treatment to packets belonging to those connections. The discovery of addresses and ports is not an easy task in the case of legacy applications. Such applications open multiple connections: TCP connections that carry control data and (usually) UDP connections used for multimedia content. Moreover, ports are usually created dynamically and at arbitrary numbers, after a negotiation between the application peers has taken place. Since their source code is not available for study, information on connections and content should be extracted in another way.

The specification of the traffic is also an important feature of a reservation request. Currently, nearly all multimedia flows are compressed, with the use of various standardized algorithms. Those compression schemes yield mainly variable bit rate traffic, which is difficult to model. Therefore, we believe that we have to rely also on real time traffic measurements. Upon the establishment of a new connection, measurements provide some first values for a resource request. With the passage of time, more accurate measurements will be available and corrections to the QoS request may be made. On the same time those measurements are displayed in a comprehensible format to the user, so that he can himself adapt the reservation.

The completion of the two aforementioned tasks is the responsibility of the Packet Filter module of the EAToolkit. This distributed module resides on the egress point of the access network and filters all incoming and outgoing packets. When a new flow is detected, the filter examines its packets, in order to extract important information: addresses and ports. Measurements are also performed in order to compute a traffic profile. Packet Filter is further analyzed in the subsequent section.

3.2 Architecture

In this section we describe the general architecture of the EAToolkit. The EAToolkit is composed of distributed objects that are physically located in various places within the access network. In the following figure the main architectural components of the EAToolkit are presented

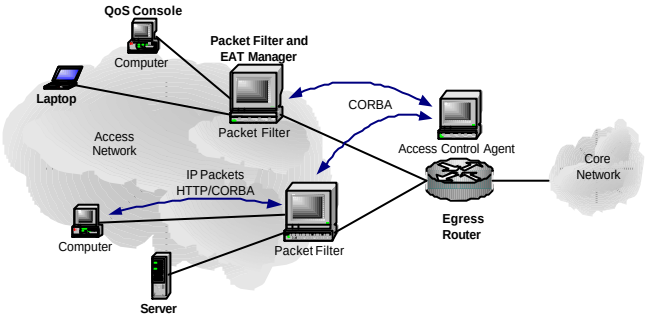

Figure 2: EATookit Architecture

3.2.1 EAT Manager

The central building block of the EAToolkit is the EAT Manager. Its main task is the coordination of the other modules in order to ensure the smooth operation of the framework. The EAT Manager may be centralized (responsible for all hosts in a sub-network) or distributed (instances of the Manager are located in every host).

The EAT Manager makes use of the concept of Session Management. A session (or flow) is defined as the set of IP packets belonging to a specific application that utilizes the quality assurance mechanisms of the EAToolkit. The discrimination of sessions is very significant in order to offer the appropriate differentiated services to different

kinds of IP flows, to compute the correct accounting information, as well as to be able to locally treat the behavior of sessions according to the QoS request. The EAT Manager therefore maintains a table with the current active sessions that have requested quality guarantees and observes the commitment to the requested levels. Moreover, it computes accounting information.

3.2.2 Packet Filter

The Packet Filter module is responsible for identifying all established network connections created by the hosts of its sub-network. The Packet Filter gathers all available information about those connections: local and remote addresses and ports, as well as traffic measurements, and forwards them to the EAT Manager.

The Packet Filter is physically located at the egress of the access network. This way, it is able to filter all packets entering and exiting, in an effort to discover new IP flows. When a new flow is detected, its addressing information is extracted and traffic measurements are carried out. Shortly, this information (preliminary, as far as measurements are concerned) is passed to the EAT Manager. The EAT Manager in turn notifies the user of a new connection and proposes QoS parameters based on the first measurements. The user may accept the reservation request as proposed by the EAToolkit, modify it, or even reject it.

In an effort to help the user reach a decision, the Packet Filter incorporates some intelligence to be used during packet processing. Most important, it should inspect the first packets of a newly detected flow. For instance, the existence of an RTP header is a positive sign of a multimedia flow. Moreover, by examining the payload format of the RTP packet, the type of media will be discovered (e.g. H.263 video or G.711 audio). Also, the Packet Filter should be aware of the use of standard ports (80 for HTTP, 544 for RTSP, 1720 for H.323), or port patterns (two successive ports may suggest two RTP and RTCP sessions).

From the above mentioned, it is evident that the user is not notified for every newly established flow, irrespective of whether it conveys control information or multimedia content. The Packet Filter has the knowledge to differentiate between flows that could actually need QoS support and those that could do well with Best Effort treatment.

An objective of the EAToolkit is also to aid RSVP applications in making a resource request to the Aquila Resource Control Layer, since RSVP is not supported in the core. RSVP messages are intercepted at the Packet Filter and their content (QoS request) is passed to the EAT Manager. The core network then forwards the RSVP messages transparently, while the EAT Manager makes the actual reservations to RCL, according to the RSVP's QoS content. In this way, the other peer receives appropriately the RSVP message for local usage, and the core network takes care of the QoS requirements, according to the provisions of the RCL.

It is evident that the Packet Filter shares some of the functionality of a Firewall. Therefore, it is expected to be equally efficient and not be a major bottleneck of the overall system. However, if scalability concerns are raised, the access network may be broken down to smaller areas. One Packet Filter may then service each segment.

3.2.3 QoS Console

The QoS Console is a graphical user interface (GUI) that is present in the form of a console on the user's desktop. Its main purpose is to present to the user, information on established connections. This information includes all the parameters supplied in the resource request sent to the RCL: addresses and ports, traffic descriptions, accounting information as well as traffic measurements. With the use of the QoS Console, the user may accept the initiation of a resource request, modify later an existing reservation, or terminate one.

4. IMPLEMENTATION ISSUES

The EAToolkit is a distributed module. Each host accessing the RCL infrastructure contains at least the QoS Console and the interface capabilities of the EAT Manager. The core EAT Manager functionality and the Packet Filter are physically located in the egress point of the access network.

In order for the EAToolkit distributed sub-modules to interface appropriately, the Java Remote Method Invocation (RMI) [7] or the Common Object Request Broker Architecture (CORBA) [8] are used. All implementations in the host are done in the Java programming language. If the implementations in the egress point are done also with Java, then Java RMI is the most suitable candidate. However, the packet filter will be probably implemented in C (for Unix). CORBA must then be utilized. Moreover, CORBA is utilized in the interface between the EAToolkit and the RCL for the transmission of QoS requests. Finally, for the user interface with the user, the QoS Console, a web browser is used with the incorporation of Java applets.

5. CONCLUSIONS

In this paper we have presented the EAToolkit, a middleware architecture for the provision of end-to-end Quality of Service to network multimedia applications. The EAToolkit resides in the access network and aids legacy non-QoS aware applications make reservation requests to the Resource Control Layer of the Aquila architecture. This is performed by automatically detecting their connection parameters (addresses, ports) and conducting local traffic measurements. The EAToolkit also provides an API for the development of new QoS aware applications that will leverage the Aquila architecture functionality.

The issue of middleware architectures for the support of Quality of Service has been also dealt with in other research projects. The Distributed Resource Controller Architecture of Columbia University [9], the OMEGA architecture of the University of Pennsylvania [10] and the QoS-A architecture of the University of Lancaster [11] are prominent examples.

Those proposals in their entirety presuppose the development of new applications that will make use of their proprietary APIs. Furthermore, in some architectures, modifications on operating systems are anticipated, in the form of callbacks for the notification of applications. On the contrary, to the proposed middleware supports applications transparently, without the necessity of any modifications. It is also our intention to shield end-users from the intricate task of selecting appropriate QoS parameters.

The middleware presented in this paper can be also utilized in conjunction with adaptive applications. The latter require modification of applications to be able to adapt to new conditions. It is not clear how this could happen with multimedia applications, where specific codecs are used. We are of the opinion that rough calculation of QoS parameters in combination with light over-provisioning should be enough, and that it is the resource reservation that should adapt (in long timescales to avoid oscillations and unnecessary communication in the Resource Control Layer), rather than the application itself.

The efficiency and performance of the proposed middleware architecture will be verified soon in field trials of the Aquila project. Moreover, user surveys will be contacted to prove the impact and acceptance of the proposed solution.

ACKNOWLEDGMENT

This work was performed in the framework of IST Project AQUILA (Adaptive Resource Control for QoS Using an IP-based Layered Architecture - IST-1999-10077 [4]) 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, D. Clark, and S. Shenker, "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, V. Jacobson and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", RFC 2638

[4] Aquila project: http://www-st.inf.tu-dresden.de/aquila/

[5] R. Callon et al., "A Framework for Multiprotocol Label Switching", Internet Draft

[6] R. Braden et al., "Resource ReSerVation Protocol (RSVP)", RFC 2205

[7] RMI, http://java.sun.com

[8] CORBA/IIOP 2.3.1 Specification, http://www.omg.org/corba/cichpter.html

[9] P. Wang, Y. Yemini, D. Florissi, J. Zinky, "A Distributed Resource Controller for QoS Applications", NOMS 2000, Hawaii, April 2000

[10] K. Nahrstedt, J. Smith, "Design, Implementation and Experiences with the OMEGA End-point Architecture", IEEE Journal on Selected Areas in Communication, Vol. 17, No. 7, September 1996, pp. 1263-1279

[11] A. Cambell, G. Coulson, D. Hutchinson, "A Quality of Service Architecture", ACM Computer Communications Review, Volume 24, Number 2, 1994, pp6-27