On providing a Dynamic QoS Management system for IP networks

L.Dimopoulou[1], P.Sampatakos[1], E.Nikolouzou[1], Y. Karadimas[2], I. S. Venieris[1]
[1]National Technical University of Athens,
Department of Electrical and Computer Engineering,
9 Heroon Polytechniou str, 157 73, Athens, Greece
Telephone: +30 10 772 2551, FAX: +30 10 772 2534
E-mail: {lila, psampa, enik}@telecom.ntua.gr, ivenieri@cc.ece.ntua.gr
[2]Q-Systems S.A.
84 Vas. Sofias Ave., 11528, Athens, Greece
Telephone: +30 10 748 9891, FAX: +30 10 779 5194
E-mail: jkar@qsystems.gr

Abstract: **This paper discusses the design of a network management system that supports QoS in IP networks. The proposed architecture, namely QMTool, uses the evolving Differentiated Services paradigm and provides end-to-end QoS for IP based applications. The novel aspects of the proposed approach rely on the integration of a collection of mechanisms including QoS provisioning for different types of traffic, flow level admission control and resource management mechanisms that interact with the routers of a network. We focus our attention on the dynamic aspects of QoS management, which is achieved by introducing a set of new technologies. The concept of the proposed management system is described in detail followed by notes of implementation issues.**

1. INTRODUCTION

The current Internet architecture does not offer the quality of service (QoS) required by the demanding multimedia applications. Those applications share the need of QoS support in order to assure that the requirements of the users are met.

The Differentiated Services (DiffServ) [1] is a proposal that aims at providing different service levels to flow aggregations. The DiffServ architecture, however, needs reliable resource provisioning in order to offer QoS support in a domain or across a number of inter-connected domains. Thus, to increase the resource utilization while avoiding violation of the service quality, traffic conditioners at the edges and mechanisms in the core of the network must be frequently reconfigured. With the equipment heterogeneity and the complex network topologies, the reconfiguration process can be a complicated task, and it can lead to severe instabilities and even more QoS violations.

Towards this end, the introduction of the QoS management [2], [3], is required to ensure that the contracted QoS is sustained. Nevertheless, management on a large-scale IP networks imposes a number of limitations. In these systems, key challenges to management solutions are handling large numbers of management resources, handling heterogeneity in equipment and technologies, as well as requirements for end-to-end quality of service. It is therefore imperative to establish a QoS-oriented network management model for supporting demanding IP applications.

Initial attempts brought mechanisms and protocols that focused on managing and configuring individual networking devices, such as the Simple Network Management Protocol (SNMP) [4]. This model worked well in early deployments of IP management systems, but now, with the evolution of QoS models such as the DiffServ framework, the complexity and overhead of operating and administrating networks is increasing enormously. As such, it is very difficult to build management systems that can cope with growing network size and complexity.

The guiding principle of this paper is to provide a simplified scalable framework for QoS management that is well aligned with the DiffServ model to provide end-to-end QoS to applications in IP networks. We present a prototype network management platform, namely Quality of Service Management Tool (QMTool). QMTool is a platform independent of the underlying technologies or vendor's specific equipment, which endows the network administrator with a set of basic functionalities for configuration, monitoring and fault management of the network elements.

Our focus is to design a management platform which provides a dynamic and scalable way for (re)-configuration of network elements, QoS provisioning for different types of traffic, management of user's profiles, creation and modification of the Network Services, as well as resource management mechanisms for re-distributing the network resources based on real resource demands. The QMTool could be considered a network management application, which performs though policy-based network management. Moreover, the integration of technologies like Extensible Markup Language (XML) [5] and Lightweight Directory Access Protocol (LDAP) [6] under a common framework, empower the design of the network management system aiming at dynamic QoS management.

Apart from the generic design, a specific implementation of the QMTool system for the management of the Aquila network [7] is provided.

The paper is structured as follows. In Section II, the functionality design of the QMTool is described. The integration of the new technologies under the common QoS framework is presented. Then, in Section III, the development of the prototype platform and its deployment in the context of the Aquila network is specified; implementation issues are given. Finally, Section IV comprises the conclusions of the paper as well as directions for future research.

2. QMTOOL FUNCTIONALITY DESIGN

2.1 Architecture

The aim of this section is to give an introduction of the architecture, in the context of which the Network management tool should be realised.

As aforementioned, the network management of today's complex and heterogeneous network environments becomes a perplexing and challenging task, so for the network administrators as well as for the service providers. By adding the intensive need for QoS features, the complexity trends to be even higher. The proposed architecture incorporates modern software engineering technologies such as Java, CORBA [8] and XML in order to facilitate the development of a unified network management platform. In other words, the diversity of the network elements and components that are subject to management is coped following an XML-based approach concerning most of the main network management functions, i.e. configuration, performance/monitoring, fault management.

The QMTool interacts with both the components of the IP layer, which is the traditional case as well as the additional components, mostly software oriented, that are introduced by the proprietary QoS control layer. It is worth mentioning that QMTool does not depend on the QoS architecture that is chosen to be realized in the managed network. In section III the appliance of QMTool in the context of the Aquila QoS architecture will be analysed.

The QMTool assists an Internet Service Provider in building services, even in co-operation with other ISPs and content providers. It handles the set of policies needed to ensure QoS in a real network like the client devices and the customers, which are allowed to make certain QoS requests. Additionally, it is responsible for the amount of bandwidth (or other QoS parameters) that can be assigned to a specific customer and is controlled by a particular server. The QMTool can be seen as belonging to a fourth layer interacting with the QoS control layer and with the IP layer if needed (see *Figure 1*). The QMTool has some similarity to a network management application. However, it performs policy-based network management and therefore it controls the operation of the underlying QoS control layer. Policies could be stored in an LDAP database.

QMTool is intended to provide the administrator with a simple, user-friendly tool for the design of the overall configuration of the network components, the monitoring of the required parameters of the network elements and the notification of a network element failure. All these functions are visualized employing an XML-based Java Graphical User Interface (GUI).

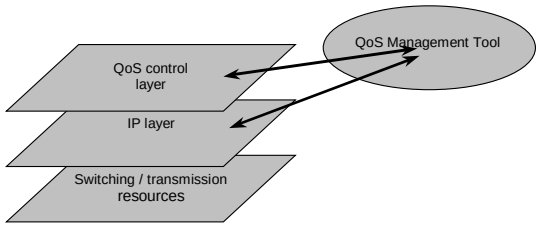

Figure 1: QoS Management Tool in a layered architecture

The architecture consists of three functional levels, as depicted in *Figure 2*, following a concept similar to the OSI model, where each level provides its services to the upper one. The lower level is responsible for the direct communication with the managed objects (e.g. routers, workstation, servers). It includes several "adapters", acting as mediator devices retaining the role to transform the proprietary parameters, which are subject to management, into XML formatted data. The second level, incorporating the XML concept, processes the data provided by the lower level (i.e. validates, stores and retrieves the data to/from the LDAP). Finally, the upper level, using the Java Architecture for XML Binding (JAXB) technology and an open source XML parser, (e.g. XMLOperator [9]) serves as a front-end tool facilitating the interaction with the network operator in a user-friendly and comprehensible manner.

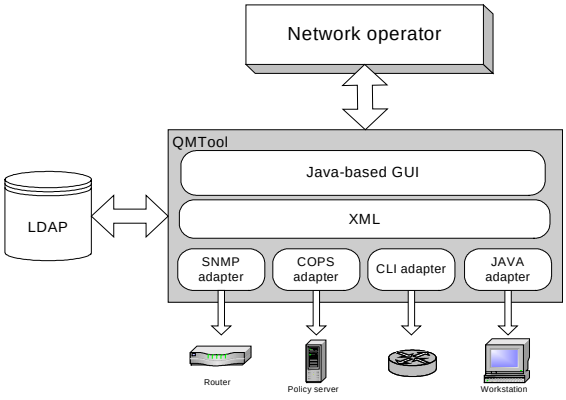

Figure 2: QMTool Structure and architecture

Adopting the aforementioned layered approach, a flexible, reliable and unified tool is developed, enabling the operator to easily manage an expandable network in which new components are regularly included.

3. DEPLOYMENT SCHEME

After having defined the architecture of a management platform, it is of great interest to see how it can be applied on a DiffServ network that is overlaid by a resource control layer for the management of its resources and for the enforcement of admission control policies. In particular, we will focus on the use of QMTool for the management of this layer and of its components although a more long-established use for the management of physical devices with already wide accepted management protocols can always be exploited.

3.1 The RCL Architecture

In brief, the architecture consists of two functional areas: the data plane that is responsible for transmitting IP packets, and an overlay control plane, namely the Resource Control Layer (RCL) 1 that is based on the Bandwidth Broker (BB) concept. Far from having a concentrated control over the network, the RCL is designated as a distributed BB consisting of several functional elements. As depicted in *Figure 3*, the three key components of the RCL are: the Resource Control Agent (RCA), the Admission Control Agent (ACA) and the End-User Application Toolkit (EAT).

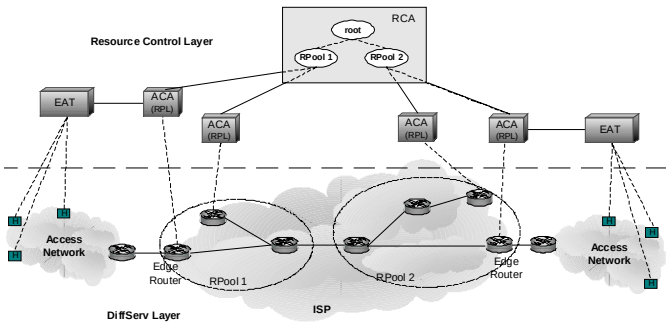

Figure 3: RCL Architecture

The Resource Control Agent is mainly responsible for the management of network resources (bandwidth) within an administrative domain. The RCA itself follows an hierarchical approach in order to handle the resources more efficiently, based on the fact that in most networks the topology follows a tree like structure. In this way, logical entities, namely the Resource Pools (RPs), are assigned the control of the resources of sub-areas of the network. The RPs are controlled by RPs belonging to higher levels of the tree hierarchy. The root of the tree is in charge of the available resources in the whole network, while each leaf of the tree structure (Resource Pool Leaf, RPL) is associated to one Admission Control Agent, which is in turn associated to one edge router (ER) of the network. The RP tree contributes to a dynamic re-distribution of resources based on real traffic demands. In this way, a dynamic QoS management is achieved though the re-configuration of the network [10].

The Admission Control Agent is mainly responsible for user authentication and authorization, reservation handling, and flow Admission Control (AC). User authentication is initially performed through an explicit login procedure, while authorization refers to the permission of a user to place a request for the selected network service. During reservation handling, flow admission control is performed. The admission control decisions are made only at the edges of the network, and consequently the corresponding RPLs are requested to ensure that the new flow can be accepted. Upon a successful reservation request, the ACAs accordingly configure the edge routers for accommodating the new flow.

Reservation requests are forwarded to the ACAs from the End-User Application Toolkit, which mediates between end-users or applications and the network. The EAT interacts with the ACA to be aware of the available network services. A reservation request specifies the flow identifiers, the selected network service and the traffic profile for it. Special support is foreseen for legacy applications as well as for end users that are not aware of traffic description details, through the use of Application Profiles. The EAT interprets the profiles and prepares the reservation on behalf of the user.

Having in mind the architecture and the functional diversity of the above-described QoS-enabled IP network, we will see in the following section how QMTool performs a uniform management of the various components belonging to the RCL.

3.2 Implementation Issues

QMTool is intended for three main tasks of network management, i.e. the configuration of the network, the fault management and the monitoring of the network elements. In particular:

A graphical interface is provided to the network administrator so as to design the topology of the network elements belonging to the RCL. Additionally, an XML Editor, integrated within QMTool, is used for setting the configuration information of these elements in the format of XML files, which are subsequently stored at an LDAP database.

A fault management module is assigned with the continuous communication of the network elements and the generation of a notification in case a failure scenario takes place. Hence, a Keep-Alive mechanism, as specified in detail in [10], is employed for that reason that is based on a simple Corba interface.

A monitoring module is entitled with the monitoring of resources (reserved and actually utilised) that are handled by the RPs pertaining to the RCA, as it has been detailed in the previous section. Here also, a uniform solution is proposed for the monitoring of data of any nature, with the use of the XML technology.

The basic functionalities of QMTool can be more clearly summarized in *Figure 4*.

[1] The reader is referred to [10] for a thorough presentation

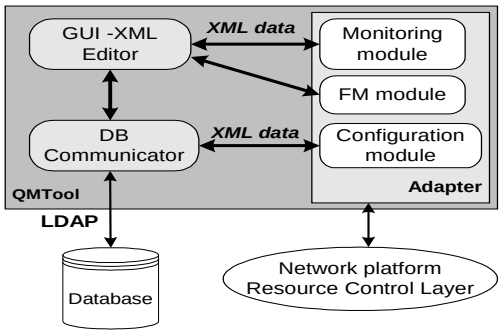

Figure 4: QMTool functional blocks

To begin with, the GUI component comprises a user-friendly interface to the administrator for constructing the RCL topology, i.e. performing the design of RPs, ACAs and EATs. Apart from the creation of a simple structure and placement of the components, an integrated XML Editor enables their configuration, with the use of predefined Document Type Definitions (DTD) that are bound to each network element. In this respect, the configuration information of each element is incorporated in simple XML files of a well known structure that are stored into the LDAP database by the Database (DB) Communicator module. It is important to note that the structure of the network can be retrieved from the database since it is hidden in the configuration files of each component. The JAXB technology is used here for facilitating the parsing of XML files and their transformation to Java objects.

Likewise, QMTool uses the XML paradigm for the creation of policies related to subscribers and to their authorised services, for the definition of Application Profiles, Network Services, Traffic Classes, for the setting of router configuration data and in general of all the information which is essential for the functioning of the examined network. As before, DTDs are always specified for each information type and the administrator produced XML files are stored into the database.

After the design and the configuration phase has been completed, the Configuration module, which is part of an Adapter being tailored to the communication means of the managed network components, serves the actual configuration of the network during its start-up procedure. In the context of the Aquila network and since we are referring to the configuration of software modules (RCL) that do not make use of the traditional management schemes (e.g. SNMP) with communication capabilities, the Configuration module and a DB Communicator is collocated with each network element. As expected, they serve the retrieval of the configuration information (XML file) from the database and its transformation to Java objects (Java Adapter).

The remaining two modules, i.e. the Fault Management (FM) and the Monitoring module, use the Corba technology for the acquisition of the required information from the network. Here also, Corba is used for providing communication means since no other conventional management scheme is employed. However, the Corba communicating parties are considered as belonging to the FM and Monitoring modules respectively and not as being part of the managed network. In more detail, all the elements that require fault detection provide a simple Corba interface to QMTool that is sufficient for examining the status of the component. As mentioned earlier, a Keep Alive mechanism is employed where the request intervals are configurable for controlling the load, although negligible, induced to the network. In case of failure, a notification (visual as well as acoustic) is produced for propagating this information to the network administrator.

As far as the monitoring is concerned, an efficient solution is proposed so as to refrain QMTool from the individualities of the monitored data of each component. The monitored data is structured to XML files of a well-known DTD. Obviously, the Monitoring module can be seen as consisting of two parts. The one, which is the main part of the Adapter, selects the data from the network elements (it is collocated with them) by being adapted to the proprietary communication means and transforms the data to XML files. In the context of the Aquila network, the JAXB technology is used for performing this adaptation. The other part simply collects the XML data (Corba communication) and forwards them to the XML Editor for being displayed. The intervals for retrieving the data to be monitored are evidently configurable since the latter can produce a significant load to the network.

In the following UML diagram (see *Figure 5*), the two Corba interfaces realizing the fault management and the monitoring functionality are depicted. To be noted that these interfaces are considered as internal QMTool interfaces belonging to the corresponding modules and not as external interfaces that need to be implemented by the managed network elements.

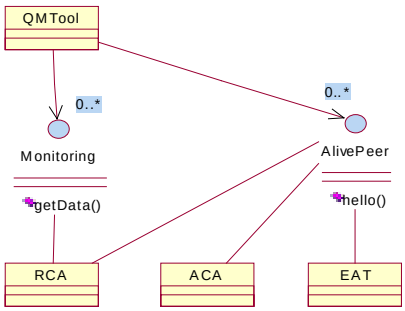

Figure 5: QMTool Corba interfaces

In the following section, we provide an example view of the QMTool revealing the GUI interface that it offers for the design and the configuration of the network.

3.3 QMTool View

In *Figure 6*, the GUI and the XML Editor components of QMTool are illustrated. In the work-area of the GUI module, the RCL topology is depicted, consisting of RPs, ACAs and EATs. The design of this network tree is enabled through the design toolbar whereas its configuration is performed with the means of an XML Editor (the open source XMLOperator is used). In particular, the XML Editor is used either for configuring the components from

scratch or for viewing (retrieval of XML data from database) an existing configuration and performing potential changes.

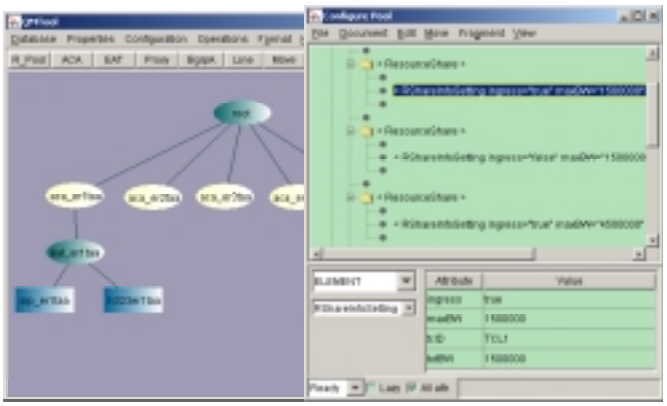

Figure 6: QMTool view

For a better understanding of the form of configuration data, an XML file is presented below (see *Figure 7*) that comprises the configuration data of a RP. The "fatherID", "poolID" and "childID" attributes/elements entail the topological information of the RP and are exploited for the retrieval of the tree structure from the database. The rest of the information, i.e. the "ResourceShare" elements, serves the configuration of the RPs with the specified resources for the several services.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RPoolSetting SYSTEM "../dtd/rpool.dtd">
<RPoolSetting fatherID="null" isLeaf="true" poolID="root">
 <ResourceShares>
  <ResourceShare>
   <RShareInfoSetting ingress="true" maxBW="1500000" tcID="tcl1"
                      totBW="1500000">
   </RShareInfoSetting>
  </ResourceShare>
  <ResourceShare>
   <RShareInfoSetting ingress="false" maxBW="1500000" tcID="tcl1"
                      totBW="1500000">
   </RShareInfoSetting>
  </ResourceShare>
  …
 </ResourceShares>
 <ChildID poolID="lrp0er1spu"></ChildID>
 <ChildID poolID="lrp0er1taa"></ChildID>
 <ChildID poolID="lrp0er2taa"></ChildID>
 <ChildID poolID="lrp0er3taa"></ChildID>
</RPoolSetting>
```

Figure 7: XML configuration file for a RP

4. CONCLUSION

In this paper, we have presented an IP QoS management system, namely QMTool. The QMTool comprises an integration of novel technologies aiming at providing dynamic QoS management features. It empowers the network administrator with a number of significant operations, like dynamic (re)-configuration of the network elements, fault management and monitoring operations. Moreover, the deployment of the introduced QoS management system to a DiffServ based network, like the Aquila layered architecture, has been presented. Implementation issues have also been covered.

As regards future work, we note that the QoS management system described in this paper has addressed only a subset of QoS management capabilities. Policy-based management issues will be in future more extensively covered. Our focus is also to study the performance of the proposed management platform in terms of exchanged signalling messages, e.g. between the database and the managed components.

5. ACKNOWLEDGEMENT

This work was performed in the framework of IST Project AQUILA [7] (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 consortium for valuable discussions.

REFERENCES

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An Architecture for Differentiated Services", RFC 2475

[2] A. Campbell, C. Aurrecoechea, L. Hauw: "Architectural Perspectives on QoS Management in Distributed Multimedia Systems", Second Workshop on Protocols for Multimedia Systems (PROMS'95), Salzburg, Austria, October 1995

[3] P.Flegkas, P.Trimintzios, G.Pavlou, "A policy-based quality of service management system for IP DiffServ networks**"**, *IEEE Network*, vol. 16, no. 2, Mar 2002 pp. 50-56

[4] Case, J., Fedor, M., Schoffstall, M., Davin, J.: A Simple Network Management Protocol (SNMP), RFC1157

[5] XML, URL: http://java.sun.com/xml/

[6] Lightweight Directory Access Protocol, URL: http://developer.netscape.com/docs/manuals/dirsdk/jsdk40/ldap.htm#1018897

[7] AQUILA Web page, URL: http:// www.ist- aquila.org/

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

[9] XML Operator, URL: http://www.xmloperator.net/

[10] M. Winter et al., "System architecture and specification for the first trial", AQUILA deliverable D1203, April 2002

[11] E. Nikolouzou, P. Sampatakos, I. S. Venieris, "An Adaptive Algorithm for Resource Management in a Differentiated Services Network", ICC2001 Helsinki