Using Fuzzy Specifications to Manage QoS

Elvis Melo Vieira **Carlos Becker Westphall**

Federal University of Santa Catarina
Post Graduate Program in Computer Science
Network and Management Laboratory
P.O. Box 476, Zip: 88040-970, Florianópolis(SC) – Brazil
E-mails: {elvis, westphal}@lrg.ufsc.br, Phone: +55 48 3317559

Abstract

This paper describes the use of fuzzy QoS specifications to manage quality of service. These specifications associate their membership functions with QoS levels perceived from a resource. Besides it is described the implementation of a QoS manager of fuzzy specifications. The QoS manager, denominated Mediator, has two parameters called α-cuts. One of them is used to allocate and the other to policy QoS requests. In short, these values determine thresholds for the QoS levels of the requests. These thresholds must be respected when the allocation and policy procedures are executed. Also, by applying the extension principle of fuzzy sets, Mediator accepts fuzzy specifications composed by two other specifications. Mediator is built as a CORBA application. Its primary purpose is to manage TCP traffic flow on IETF Differentiated Services Platforms.

Keywords: *Qos Management, QoS Specification, Distributed Object-Based Applications, Internet Differentiated Services.*

1. Introduction

Nowadays there are several QoS frameworks such as ATM [1], OSI [2], TINA [3], IETF IntServ [4] and IETF DiffServ [5]. Through these frameworks, an application can inform a QoS specification to allocate a resource such as link bandwidth, delay jitter, packet loss and so on. All them don't accept specifications with imprecise values for resource allocation. Their specifications must have precise values or crisp numbers. For example, to allocate an ATM virtual circuit [1], it is necessary to indicate certain parameters, such as cell or loss rate. These parameters do not indicate other possible values to be allocated if the network cannot delivery the cell or loss rate requested. The application could request a cell rate of 1,000 cells/s for a quality level of 100%, as well as a rate of 800 cell/s, with 80% of QoS level, and so on. Even a continuos function could be specified associating cell rate and QoS level.

Making QoS allocations with crisp numbers does not allow policy specifications, such as "allocate a service if there is a good discard packet" or "shutdown a service if the packet error rate is very bad". These are subjective ideas. "Good" or "very bad" specify classes of possible values to discard or packet error rate.

The item 2 of this paper presents what are the fuzzy QoS specifications proposed. This item describes the meaning of a fuzzy specification and how it can be used. The item 3 describes the Mediator's architectural model. Its modules and services are discussed with details. The item 4 shows two examples to demonstrate the applicability of the fuzzy specifications. In these examples, Mediator is used to manage delay and delay jitter of TCP

Connections. The item 5 presents some conclusions about this work and discusses some future extensions. Some references are given in the item 6 to get more details about some correlated works.

2. QoS Specifications

2.1 QoS level

QoS level is defined, in this work, as a metric of quality. It informs the quality of a service being delivered to an application. The value that a QoS level can assume is in the interval [0,1]. The value 0 means that the service delivers no quality at all, but 1 means that the maximum quality is being delivered.

QoS Level is associated with memberships functions of fuzzy specifications, as it will describe. So QoS level and memberships of a fuzzy specification are considered as the same in this work and both terms are used.

2.2 Fuzzy QoS Specification

A QoS fuzzy specification indicates the value of a QoS parameter (p) by a fuzzy set (see appendix). Instead of a unique value that the application requests, there a set of possible values m for a metric M. Each value m has a QoS level associated which is determined by the membership function $\mu_M(m)$ of the specification's fuzzy set.

The graph in Figure 1 determines a QoS level (l) given by the following function:

$$l = 1 - 30t \qquad \text{(I)}$$

Where t is the delay (to send a packet of 1024 bytes) in seconds (s).

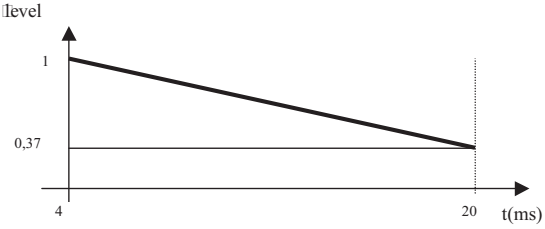

Figure 1 - Graph for equation (I)

Therefore, the application indicates a range of values m that the QoS manager can use to allocate a request. It can be observed two important consequences:

- Through a fuzzy specification, a application informs a range of values acceptable with different QoS level, not only one value. Therefore, the decision what QoS level the application will have is determined by the QoS manager. Besides, the negotiation process between an application and a QoS manager is shorted because they do not need negotiate other values anymore. The possible values are already determined by the fuzzy specification.
- QoS contracts using fuzzy specifications are different. They don't have thresholds values, but membership functions. However, if a threshold value is necessary, it has to be determined by its membership functions. In Figure 1, the value 20ms is a threshold value. When the value for delay is greater than 20ms, the Qos level is considered zero.

2.3 Fuzzy QoS Specifications With More Than One Metric

Fuzzy QoS specifications can have more than one fuzzy metric. Let $M_1,..,M_r$ be a set of fuzzy metrics in $X_1,..,X_r$ universes respectively, and X the Cartesian product of this universes, i.e., $X = X_1 \times .... \times X_r$. Also, let f a mapping from $X = X_1 \times .... \times X_r$ to a universe Y like $y=f(x_1,...,x_r)$. The extension principle (see appendix) allows to induce another fuzzy set C on Y such that

$$C = f(M_1,...., M_r) = \int_{X_1 \times .. \times X_r} \min(\mu_{M_1}(x_1),..., \mu_{M_r}(x_r)) / f(x_1,..., x_r) \qquad (II)$$

The Figure 2 shows an example. In this figure, the networks net1 and net2 are linked by two paths p_1 and p_2:

p_1 = (router1, router3,router5)
p_2=(router1,router2,router4,router6)

The Table1 shows the links delay in a determined instant. This table shows the delay, desired delay and QoS level on each link of the network in Figure 2. The QoS Level is calculated by:

QoS Level = 1 - (delay - desired delay)/desired delay if delay > desired delay
QoS Level = 0 if delay ≤ desired delay.

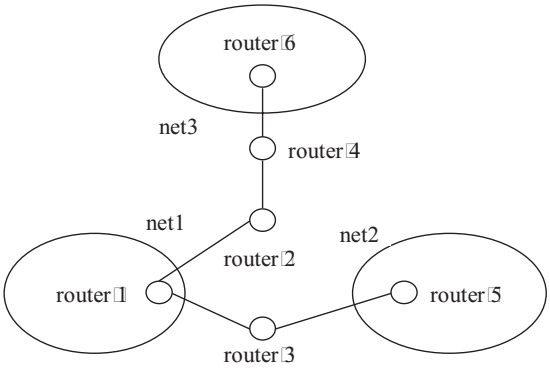

Figure 2 - WAN composed by three remote networks

Then the delay on each link (i,j) can be considered as a fuzzy set D_{ij} in a universe X_i (integers greater than 0) given by (the second "/" has the fuzzy set meaning - see appendix):

$D_{ij}(delay) = 1 - [(delay - desired\ delay)/desired\ delay]/delay$

For example, to link (router1,router2), i = 1 and j = 2, is associated with $D_{12} = 1/10$. But, the link (router1,router3), i = 1 and j = 3, is associated with $D_{13} = 0,33/20$.

The QoS level of the path p_1 and p_2 can be determined by using the Extension Principle and recalling that delay is an additive metric [9]:

$P_1 = D_{13} + D_{35} = min(0.33,0.66)/(20 + 20) = 0.33/40$
$P_2 = D_{12} + D_{24} + D_{46} = min(1,0.75,0.50)/(10 + 15 + 30) = 0.50/55$

Where P_1 and P_2 are the fuzzy metrics of the links p_1 and p_2 respectively. Although the delay of p_1 is smaller than in p_2, p_1 has a smaller QoS level. Finally, a QoS level for router

can be determined considering only its paths. For example, the router1 has a QoS level of min(0.33,0.50) = 0.33.

Link	Delay	Desired Delay	QoS Level
(router1,router2)	10ms	<12ms	1.00
(router2,router4)	15ms	<12ms	0.75
(router1,router3)	20ms	<12ms	0.33
(router3,router5)	20ms	<15ms	0.66
(router4,router6)	30ms	<20ms	0.50

Table 1 – Link Delays

3. Mediator – A System to Manage Fuzzy QoS Specifications

The Mediator is a system to manage fuzzy QoS specifications. Initially, the Mediator was built to manage TCP applications by using the IETF DiffServ Framework [5]. However, it can be extended to manage UDP applications and to use other frameworks like IETF IntServ [4].

3.1 Concepts Used by Mediator from Fuzzy Sets

3.1.1 α cuts

Mediator uses the α cut (see appendix) concept to allocate and policy an application request about a metric M.

$$M_\alpha = \{ m \in M, \mu_M(m) \geq \alpha \} \qquad (III)$$

It has two α cuts defined:

- **Allocation α cut** – A QoS request to be allocated needs that its QoS level fits into α cut determined by the allocation α cut parameter. This parameter is called minimum level in Mediator. It is static, i. e., it doesn't changed after the Mediator starts.
- **Policing α cut** – The policing method tries to keep all applications into an α cut determined by the policing α cut parameter. This parameter is called current level in Mediator. It is dynamic, i. e., it changes in each policy iteration. The effect of this it is the tendency of all requests to presents QoS levels almost equal. However, the current level cannot be less than the minimum level.

3.1.2 Extension Principle

Mediator allows defining QoS requests that have one or two fuzzy QoS metrics. To calculate the resulting QoS level (membership) of a fuzzy specification composed by two other fuzzy specifications is used the extension principle. Then an application can inform a request that specify the delay, jitter delay, or both delay and delay jitter of a TCP connection.

3.2 Architecture's Overview

The Mediator's modules are illustrated in Figure 3. As it can see, Mediator is a distributed system. It has two types of servers: The Policy Decision Point – PDP and a Policy Enforcement Point – PEP. The PDP is the QoS manager, responsible for policing all QoS requests. PEPs map PDP adjust messages to specific commands that control DiffServ configuration in each node. There is only a PDP server, but one PEP server for each IP node that supports DiffServ. This can be illustrated by the example shown in figure 6.

Applications specify QoS parameters for their TCP connections through fuzzy QoS specifications. These specifications are implemented by FlowProfile objects and stored in the PDP's repository. For each request in repository, the PDP measures its flow delay. Then PDP sends adjust messages to PEPs for making changes in DiffServ parameters, if necessary.

For the time being, Mediator only accepts delay specifications of TCP Connections. They have to specify delays for sending packets of 1024 bytes into the TCP connections. But Mediator is being changed to accept other sizes of packets too.

3.3 Policy Decision Point - PDP

The PDP is responsible for allocation and policing of QoS requests according to their fuzzy specifications. To do so, the PDP has services for request policing, Qos routing and request allocation. These services are implemented by CORBA methods and they are explained shortly below along with some PDP attributes.

3.3.1 Policing Service

The PDP has an attribute called current level. This value is a minimum QoS level that the PDP tries to keep for all QoS requests. Basically, for each one, the PDP measures it and applies the specification membership function to the value taken. The result is compared with current level. If it is smaller, an adjustment is done for the request that has the highest QoS level and whose traffic flow intercepts the flow being managed. However, if the highest QoS level is smaller than the current level, then the current level is decreased by a step and it is tried again. The Adjust service (explained later) is internal to PDP, i. e., it is not available for applications.

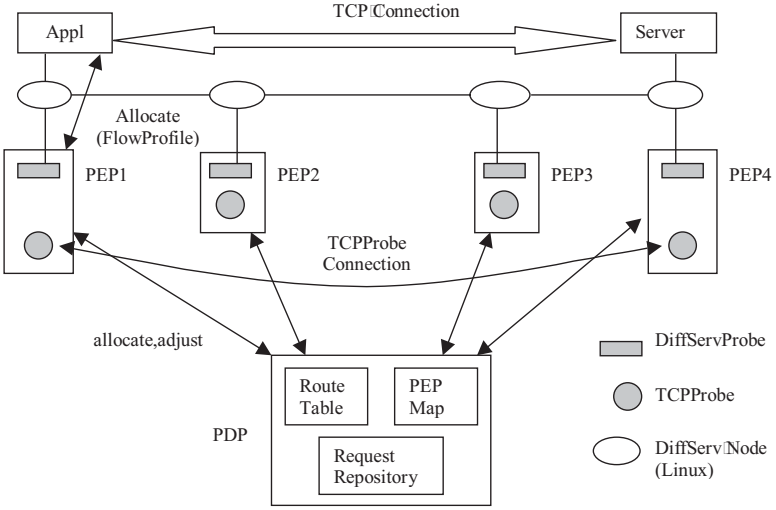

Figure 3 - Mediator Modules

3.3.2 Routing Service

The Route Table has routes to find PEPs over the network. As a PEP starts, it registers itself in the PDP by calling the PDP method RegisterPEP. This method takes the addresses for each PEP probe and adds entries on the RouteTable. Through the RouteTable, the PDP can discover all PEPs involved in a TCP connection.

Figure 4 shows the routing process. Each PEPs (pep1, pep2, and pep3) has DiffServ probes. pep1 has the probe (192.168.1.1,192.168.1.2), pep2 has the probes (192.168.1.2, 192.168.1.1) and (192.168.2.1, 192.168.2.2) and pep3, the probe (192.168.2.2, 192.168.2.1). For each address, the Route Table builds its shortest path tree. So that, it is simple to calculate a route containing all PEPs that a TCP connection passes. For example, to calculate the route between 192.168.1.1 and 192.168.2.2, it is sufficient to use the tree for address 192.168.1.1 and to find out a route containing entries. In this case, the resulting route is ((192.168.1.1,1), (192.168.1.2,0), (192.168.2.1,1), (192.168.2.2,0)).

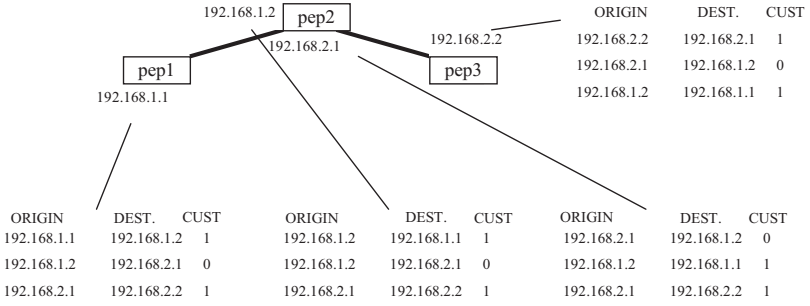

Figure 4 - Routing Process of the Mediator

There are also some works using fuzzy sets to deal with QoS routing [7] [8] [9][10] [11]. In [7] it is presented an approach that tries to optimize the bandwidth utilization of the links and propagation delay of the services. The results of some simulations are also showed in [7].

3.3.3 Allocation Service

To allocate QoS for a request, the method Allocate of the PDP can be used. It takes a FuzzyProfile object and applies the routing procedure previously described. Then it does the following steps:

1. Take the value of the metric considered between source and destination address.
2. Apply the membership function to this value to determine the available QoS level .
3. If the available QoS level is smaller than the minimum level then adjusts are necessary. After adjusts, new try to allocation is done.
4. Otherwise allocate OK.

If no allocation is got until a certain number of adjusts, then the allocation procedure is considered as failed. The minimum level parameter is fixed when the PDP server starts. It does not change during the lifetime of the Mediator.

3.3.4 Adjust Service

Adjust of application's QoS parameters are carried out by the internal method Adjust. Not only the Policing Service, but also the Allocation Service needs this service. In short, the adjust process has the following steps:

1. Find the target route between destination and source addresses of the request (in FuzzyProfile object).
2. Find the maximum QoS level among requests whose route intersects the target route.
3. If the maximum QoS level minus a step level is greater than α, then decrease the QoS level for this application and try again.
4. Otherwise, there is a QoS violation.

The parameter α is the minimum level if the Adjust is called by the Allocate method. But it is the current level if the Adjust is called by the Policy method. The Figure 5 shows a Mediator's partial log of the delays and QoS levels of the requests. Also the current level is shown. For this experience, the Mediator was started with the minimum level set up to 0.5.

The first column indicates which request the row is referring to. The second one indicates the delay measured and the third one, the QoS level given by the fuzzy QoS specifications. Lastly, the fourth column indicates the current level.

r=0	0.001514	0.95458	0.77524
r=1	0.001555	0.9378	0.77524
r=0	0.001561	0.95317	0.91522
r=1	0.005721	0.77116	0.91522
r=1	0.005388	0.78448	0.77364
r=0	0.00158	0.9526	0.93456
r=1	0.001545	0.9382	0.93456
r=0	0.001585	0.95245	0.90103
r=1	0.005827	0.76692	0.90103
r=1	0.002018	0.91928	0.85372

Figure 5 - Log of The Adjust Procedure

It is observed that the second request appears with some repeated rows. This mean that an adjustment in the QoS parameters was done for the request that has the highest QoS level. For example, the second sequence shows that the first request has the QoS level equal to 0.95317 while the current level was 0.91522. Then it was in the policy α cut. But the second request does not because its QoS level is 0.77116. Then an adjustment was done. The first adjust results a current level of 0.77364 and the Qos level of the second request is 0.78448. Then one more adjusts were done because the second request is in the policing α cut too.

3.3.5 PEPMap

This table contains one entry for each probe and it is up to PEPs to register them on PDP. In fact, once a PEP asks a PDP for registration, the PDP gets all PEP's probes and puts their source and destination addresses into the PEPMap. Also, a reference to the PEP is put into.

The PEPMap is necessary for locating a PEP, provide that the source and destination addresses for one of its probes is known. The allocate method uses this table intensively. In fact, once the TCP connection address is given, a route is searched in the Route Table.

Afterwards, the PEP object references are taken from the PEPMap using the route found. For the route in the example before, the PEP objects found are pep1, pep2 and pep3.

3.3.6 QoS Request and Repository Database

A QoS request is a structure that contains information about a request for QoS. All requests are maintained in the PDP's Repository database. Once an application creates a FlowProfile object, it calls the PDP method allocate with the new specification created. Then a new QoS request is created into the Repository's database and returned to the application.

3.4 FlowProfiles

A traffic flow is specified using a class derived from the abstract class FlowProfile. In fact, FlowProfile specifies two methods that must be implemented by the derived class: membership and value. The membership method is the implementation of the membership function for the fuzzy specification. The method value is the reverse of the membership function, which some PDP's methods require.

So, PDP acts as a factory of FlowProfile objects. To do so, PDP offers certain methods. Up to now, there are only two methods: createLinearFlowProfile and createCompLinearFlowProfile. Each one creates FlowProfile objects with a specify membership function. The PDP createLinearFlowProfile method creates a FlowProfile object that has a linear function (y=a+bx) as its membership function. The createCompLinearFlowProfile method creates a FlowProfile object composed by two other FlowProfile objects.

3.5 Policy Enforcement Point - PEP

The PEPs are responsible for translating the PDP actions into actions on DiffServ nodes. Like PDPs, PEPs also have some attributes and services. The most important ones are explained below. They are probes that are responsible for two kinds of operations:

1. Taking measures of TCP flow delay.
2. Making adjusts on a DiffServ node configuration.

On whole, probes are abstraction of low levels and they depend on what metric is being used. In addition to probes, DiffServProfile objects are also necessary. These objects are used to map FlowProfile to a profile of traffic in the DiffServ node.

3.5.1 DiffServProfile

This class aims to capture all characteristics necessary to define profiles of traffic in DiffServ nodes. Objects from it are created by PEPs to map the allocation requests from PDP. After that, they are sent to DiffServProbes that take and translate them to commands that are understandable to DiffServ devices.

The most important attributes in each DiffServProfile class are:

- DSCP: Indicates the PHB in which the FlowProfile in PDP was mapped. It can be AF11, AF21, AF31 and AF41. Presently, neither PHB EF nor best effort (BE) is allowed
- Source and destination addresses: these addresses indicate the source and destination of the TCP connection being monitored.
- Source and destination ports: these addresses indicate the source and destination ports of the TCP connection being monitored.
- Bandwidth: Indicates the bandwidth reserved for this profile.
- Token Bucket Size: Indicates the token bucket size reserved for this profile

3.5.2 DiffServProbes

DiffServProbes take the DiffServProfiles and create the corresponding configurations on DiffServ devices. All DiffServ probes are derived from the DiffServProbe class. This class is abstract because it has four virtual methods: SetDeviceOnEdgeNode, LoadProfileOnEdgeNode, SetDeviceOnCoreNode and LoadProfileOnCoreNode. The derived classes must implement these methods. SetDeviceOnEdgeNode and SetDeviceOnCoreNode methods initialize DiffServ devices on edge and core nodes respectively. Meanwhile, the LoadProfileOnEdgeNode method is used to load a DiffServ profile on DiffServ edge nodes and LoadProfileOnCoreNode method, on core nodes.

Each DiffServ probe is identified by two IP addresses. For example, pep2 shows in Figure 4 has two probes identified by (192.168.1.2,192.168.1.1) and (192.168.2.1,192.168.2.2). Every time that a PEP is registered, the addresses of all its probes are copied to PEPMap on PDP. Also, entries in the Route Table are created.

For the time being, only the class LinuxDSProbe, derived from DiffServProbe, is implemented. This class handles DiffServ devices on Linux Systems. However, another class being investigated is LDAPDSProbe for systems that take DiffServ policies from LDAP databases like some routers and servers.

3.5.3 TCPProbes

TCP Probes measure TCP connection delays. Because there is no direct way to get the delay of a TCP connection, an almost intrusive method is used. In fact, it introduces a TCP message to get the delay, which could slightly fake the measure. However, the TCP connection used is different from the connection being controlled. The **Figure 6** describes this situation. The delay of the connection (Application, PC, Server, PS) between server and client is measured by a TCP probe connection (Application, M, Server, N).

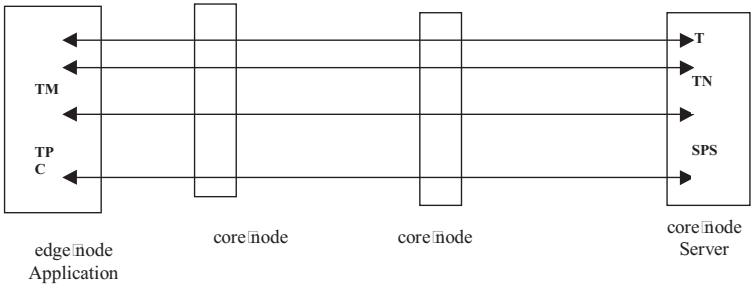

Figure 6 Probe Connections

3.5.4 PEP Initialization

When a PEP starts, it still has no DiffServProbe. Each probe to be added has to be indicated by PEP method AddProbe. When the PEP asks for registration in PDP, all probes added before are mapped into PDP (PEPMap).

3.6 Mediator's Platform Supporting

For now, only CORBA applications can request QoS to PDP. Through the Mediator's idl specification, applications can be built using any language, which has an IDL mapping. PEP, PDP and FlowProfile are CORBA objects. To build them, it was used the ORBACUS

3.3.1 for C++ and GNU GCC 2.95 C++ Compiler. Up to now, only Linux (kernel 2.2.10) nodes that have the Almesberger's DiffServ implementation [12] are supported.

4. Examples of Mediator's Utilization

4.1 Example 1

The first example is an application that has two TCP connections to be managed. The Figure 7 illustrates the environment where the example takes place. As it can see, the TCP connections have the source address 192.168.2.2 and 192.168.1.1 as their destination. The source and destination ports for the first connection are 5501 and 5502, respectively. For the second connection, they are 5503 and 5504. Also, the traffic flow considered is unidirectional, from source to destination (shown by bold lines). For the inverse direction it is necessary to allocate another QoS request.

The first connection has the specification given by the equation (I), which was described before. The second connection has also a linear function $y = 1,0 - 0t$. It must be made clear that the applications do not need to be modified to have their traffic flows monitored by the PDP. Another application can be built that asks for QoS request on behalf of the unchanged applications. In the example shown, the PDP Application does the QoS requests. Finally, there are others TCP connections such as the CORBA ORB, telnet and NFS, but they are not considered.

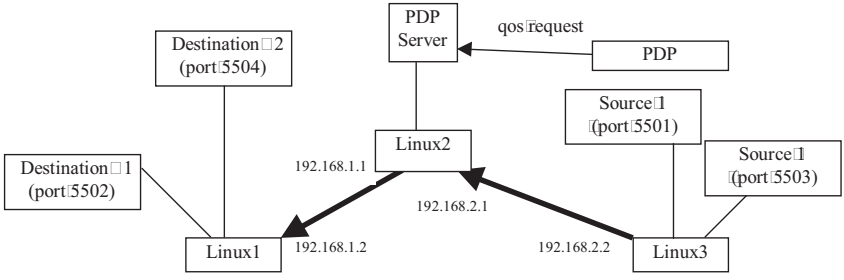

Figure 7 Example Environment

The Figure 8 shows a graph containing measures delays of the two connections being managed. Also, a test connection that does not ask a QoS request is shown too. It can be seen that for the connections managed by the Mediator, the connection delays vary a little, always below 0.0025s. However, the test connection has delays that vary a lot. Its values are between 0.0115s and 0.0043s. It could be concluded that Mediator keeping the QoS level of the requests, got their delays almost constant. The traffic of others applications didn't interfere with the two connections being monitored.

The Figure 9 shows the QoS levels measured. It can observed that the current level is the same of the QoS level of the first request most of the time. Also can be observed that the second request has an almost constant line of QoS Level. It is due to the linear function of its fuzzy specification. It has a linear coefficient smaller than the second request.

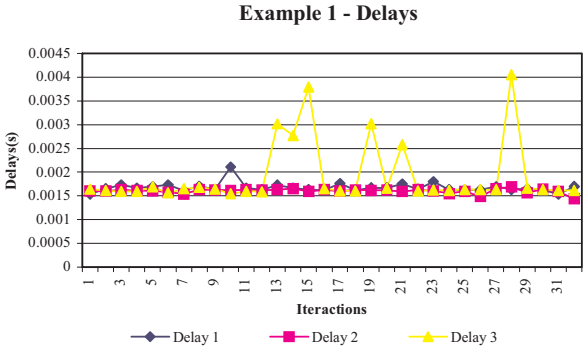

Figure 8 - TCP Delays of The Example 1

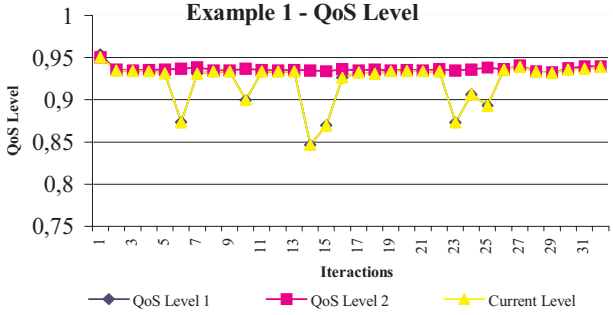

Figure 9 - QoS Levels of The Example 1

5. Conclusions

By specifying QoS requests through fuzzy specifications, it is possible to indicate how the QoS level will be, according to the measures taken from determined metrics. With these specifications, the QoS manager can policy all traffic flows and determine the QoS level for each application. The QoS manager only allocates QoS requests whose QoS level is smaller than the allocation α-cut parameter. Through a policing α-cut parameter, the QoS manager tries to keep all QoS levels greater than or equal to the value α calculated in each iterations.

The Mediator system is being built to test the utilization of fuzzy specifications and some details of this implementation were described here. As a result, an example of a program that uses a fuzzy QoS specification was showed. By this example, it may see that the specification determines how the QoS level depends on the metric considered. This is different from the traditional approach based on what QoS metric is desirable. This new

approach allows to Mediator to determine the application QoS levels that will take place, instead of being determined by the applications themselves.

Because of managing TCP connections, Mediator is very good to manage applications like Distributed Database Management Systems or High Performance Computation. But, Mediator is being extended. For now, two works are being done: The first one is to extend the Mediator to manage UDP connections. The second one is the work in [6] to add a more robust routing scheme. With these two inclusions, Mediator can be used to manage QoS of Distributed Multimedia Applications.

6. References

[1] M. de Prycker. Asynchronous Transfer Mode: Solution for Broadband ISDN. 2nd Ed., Ellis Horwood, New York, 1993.

[2] ITU T, Information Technology – Quality of Service: Framework, ITU T Recommendation X.641, ISSO/IEC IS 13236, December 1997.

[3] TINA C, Overall Concepts and Principles of TINA, TB__MDC.018_1.0_94, February 1995.

[4] RFC 2212; S. Shenker, Specification of Guaranteed Quality of Service, IETF, September 1997.

[5] RFC2475; S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, An Architecture for Differentiated Services, IETF, December 1998.

[6] D. Waddington, G. Coulson, D. Hutchison, Specifying QoS for Multimedia Communications within Distributed Programming Environments, Proceedings of the Third International COST237 Workshop, November 1996.

[7] E. Aboelela, C. Douligeris, Fuzzy Generalized Network Approach for Solving an Optimization Model for Routing in B $SDN, ECE University of Miami.

[8] M. Sakawa, Fuzzy Sets and Interactive Multiobjective Optimization, Plenum Press, NY, 1993.

[9] Z. Wang and K. Crowcrof, Quality of Service Routing for Supporting Multimedia Applications, IEEE Journal on Select Areas in Communications 14, pages 1228 1234, 1996.

[10] R. Vogel, R. G. Herrtwich, W. Kalfa, H. Witting, L. C. Wolf, QoS Based Routing of Multimedia Streams in Computer Networks, IEEE Journal on Selected Areas in Communications 14, pages 1235 1244, 1996.

[11] J. K. Chemouil, M. Lebourges, A Fuzzy Control Approach for Adaptive Traffic Routing, IEEE Communications Magazine 33, pages 70 76, 1995.

[12] Differentiated Services on Linux, Almesberger, W., Salim, J. H., Kuznetsov, A., Internet Draft draft almesberger wajhak diffserv linux 01.txt, May 1999.

[13] D. Dubois, H. Prade, Fuzzy Sets and Systems Theory and Applications, Academic Press, New York, pages 9 20, 1980.

[14] Assured Forwarding PHB Group, Heinanen J., Baker F., Weiss W., Wroclawski J., RFC2597.