# Converting QoS policy specification into fuzzy logic parameters [*]

Marcial P. Fernandez[a],Aloysio de C.P. Pedroza[b] and José F. Rezende[b]
mfernandez@ic.uff.br, aloysio@gta.ufrj.br, rezende@gta.ufrj.br

[a]Instituto de Computação - Universidade Federal Fluminense (UFF)
Rua Passo da Pátria, 156 - Bloco E - 3º andar - CEP 24.210-240 - Niterói - RJ - Brasil

[b]Grupo Teleinformática Automação(GTA) - Universidade Federal Rio de Janeiro(UFRJ)
COPPE/PEE - C.P. 68504 - CEP 21945-970 - Rio de Janeiro - RJ - Brasil
Departamento de Eletrônica/EE-UFRJ

The policy based management is a technique to coordinate the configuration of several equipments in a network based on Service Level Agreements(SLAs). These abstract policies are difficult to be interpreted and implemented by network equipments, that requires absolute values, and fuzzy logic has been showed to be efficient to represent abstract values. A fuzzy controller should implement a dynamic provisioning mechanism used to reconfigure all nodes according ingress traffic. This work proposes a methodology to map policies in fuzzy parameters to achieve the desired QoS in a DiffServ domain. The functionality are demonstrated by simulation of an IP Telephony application crossing a DiffServ domain.

## 1. INTRODUCTION

Differentiated Services (DiffServ) [ 1] is a proposal that aims at providing Quality of Service (QoS) for a certain number of service classes in the Internet. This is achieved through service discrimination among flow aggregations pertaining to these classes. The DiffServ (DS) architecture is scalable, offering guarantees for different classes, however the QoS can suffer violations when there are traffic jam in the aggregation of several flows in the same class, compromising edge-to-edge QoS.

This scalability has its price: we cannot guarantee the QoS for all flows at the same class. As Internet has an uncertain traffic estimation, we should be prepared to detect QoS violations. Solutions as IntServ make possible the control individually for each flow, guaranteeing QoS for them, however, IntServ has scalability problems in core routers.

The necessity of QoS guarantees in Internet lead us to use of dynamic provisioning mechanisms. The unpredictability and the randomness of the traffic entering in a DS domain forces the use of some dynamic reconfiguration technique. But because the complexity of those mechanisms, most of the network operators prefer over-provisioning the resources to obtain the desired QoS. This approach, however, presents high costs, as the full capacity is not used all the time.

[*]This work was supported by UFRJ, CNPq, CAPES and ANATEL

In previous works, we proposed a fuzzy controller that reconfigures nodes dynamically, adjusting network provisioning according the incoming traffic. This proposal showed efficient to improve QoS in a simple DS domain[ 2] and in a complex domain with different topologies with 40 nodes[ 3]. The fuzzy approach is justified by the non linearity and absence of a mathematical model to estimate the traffic[ 4]. Compared to a conventional controller PD (Proportional Derivative), the fuzzy controller shows significant advantages to treat imprecise variables. To improve the fuzzy controller efficiency we used techniques based on genetic algorithms (GAs) to optimize the fuzzy controller's parameters[ 5, 6].

The policy based management[ 7] has been showed an effective technique to coordinate the configuration of network equipments to obtain the desired QoS. Most of the works on policy based management focus in the policy specification and in information model where the policies will be applied and little work has been made on how entities execute the policies and how to choose configuration commands in actual equipments. Besides, there is little study about application of policies in large environments, where different interpretations of the same policy can cause instabilities and flaws in network operation[ 8].

This work proposes an architecture of a policy based management system to coordinate the dynamic provisioning in a DiffServ domain. We show a proposal to map policies in fuzzy controller parameters, used to reconfigure the provisioning. To validate the methodology a controller was developed using this methodology. So, a simulation of a prototype was achieved, evaluating delay, jitter and drops of IP telephony application. Afterwards, we show an evaluation of fuzzy controller variating some parameters.

This paper is organized as follows: section 2 presents the system architecture and methodology of controller development; section 3 shows the policy mapping methodology; section 4 shows the prototype implementation following the proposed methodology; section 5 shows the simulation results; and finally, the section section 6 presents the conclusions of this work and suggestions for future works.

## 2. SYSTEM ARCHITECTURE AND METHODOLOGY

The objective of proposed system is coordinate the network equipment configuration to achieve required QoS, maintaining the scalability of DS architecture. Then, we propose a controller that implements the dynamic provisioning in the nodes from the policies specified by the network operator. Thus, we introduced the methodology for controller development.

### 2.1. System architecture

The architecture goal is to achieve QoS garantee maintaining system scalability. All edge and core nodes in domain have a controller that measures the current state, calculates the new configuration using a fuzzy logic mechanism, and execute the configuration command in the node. Since fuzzy logic processing is not heavy, it can be executed in each node without interfering in router performance, even it has a interval operation time of seconds. Besides, the node is responsible for collecting the equipment state information (e.g.,delay in the queues) and report to the manager, using a management protocol, for instance SNMP (*Simple Network Management Protocol*).

Some decisions require the knowledge of domain state, when the policy manager reconfigures all the necessary nodes. An example of this situation is when the delay in the priority class in network core increases and there is no more resources to allocate, forcing a reduction in the input rate in edge nodes to avoid drops in core nodes.

The policy manager, unique in domain, collect all node information. It makes the optimization with genetic algorithm (GA) and reconfigures all the nodes regularly, using COPS protocol (*Common Open Policy Service*), for instance. Since the GA requires large amount of computer resources, that could interfere in the routers performance, it is executed in policy manager. We also remind that the GA optimization occurs at larger intervals, in order of hours or days, therefore the scalability can be maintained even for large domains.

### 2.2. System development methodology

We show, in figure 1, the proposed methodology flowchart. In this figure, the rectangles represent a methodology procedure and the parallelograms represent the interactions of system with users or network equipments. The network operators should accomplish QoS metrics specified in SLA agreements, for instance, the maximum delay in network is 100 ms. The highlighted boxes will be detailled below.

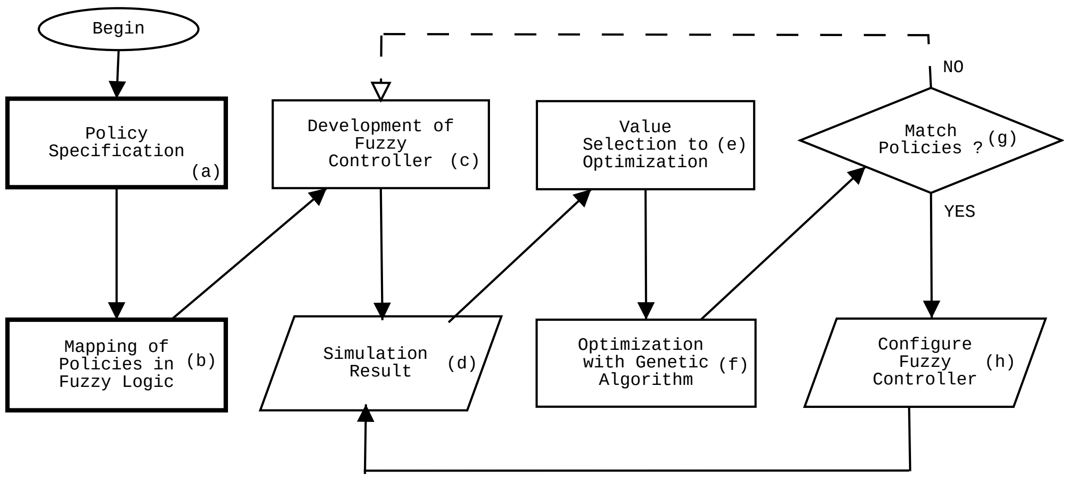

Begin
Policy Specification (a)
Mapping of Policies in Fuzzy Logic (b)
Development of Fuzzy Controller (c)
Simulation Result (d)
Value Selection to Optimization (e)
Optimization with Genetic Algorithm (f)
Match Policies ? (g)
NO
YES
Configure Fuzzy Controller (h)

Figure 1. Proposed methodology flowchart.

The first stage is the specifications of administrative policies, based on SLAs requirements, shown in figure 1(a) and detailed in section 3.2. The following stage, figure 1(b), shows the controller configuration from administrative policies, for this, a mapping methodology of policies in fuzzy parameters is proposed in section 3.3.

The next stage is the fuzzy controller development, showed in figure 1(c). This stage, detailed in [ 2], defines the fuzzy controller parameters, as membership functions and rules base, based on the policy specification mapped in previous stage. The controller can be configured on network equipments, or as in our experiment, running a simulation. Next stage, showed in figure 1(d), collect the input and output values of fuzzy controller and performance metrics, for instance, delay and packets drop.

With those values, we can choose all the combinations that maximize the performance metrics, e.g., smallest delay and drop rate, showed in figure 1(e). The following stage, shown in figure 1(f), is the fuzzy controller parameters optimization with GA, using as objective function the values selected in the previous stage. The stages (d), (e) and (f) were detailed in [ 9].

The GA optimization can distort the membership functions and rule base, making the controller output not accomplishing the original policies. In stage 1(g), the new membership functions and rules are evaluated to test the result against policy specification and, in case it disagree, the fuzzy controller needs to be redefined in stage 1(c), showed with a dashed line. If the controller produces a coherent result with original policies, it can be used in the equipment for normal operation, as showed in stage 1(h).

The methodology considers optimization process running continually, adapting the fuzzy controllers according network changes (link activation or deactivation, change in traffic pattern, etc.). As those changes are usually small and eventual, the GA optimization is efficient. The adaptation process is shown in the figure 1 with a solid line, considering that administrative policies keep unchanged. Otherwise, it should begin in from initial stage (figure 1(a)).

## 3. IMPLEMENTING POLICY MANAGEMENT

The policy based management[ 7] has been used to coordinate the configuration of network equipment to obtain the desired QoS. We can define a **policy** as a *rule that defines a choice in the behavior of a system*[ 10]. To specify the policies we used Ponder[ 11] language, presented as follows.

### 3.1. Language of specification of policies: Ponder

The *Ponder* language was proposed by Damianou *et al.*[ 10] to specify management policies textually, according Sloman [ 7] and Lupu [ 12] proposals. It is a declarative object oriented language and offers to users a simple way to specify policies.

### 3.2. Specification of QoS policies

We can specify the management policy from administrative requirements. In our example, we defined a policy that all priority should be given to the EF *(Expedited Forwarding)* class, i.e., the BE*(BE - Best Effort)* class will be reduced whenever there is a reduction in the EF QoS metrics. We defined two controllers: one for the scheduler, placed in all nodes, and one for the conditioner, placed just in the edge nodes.

### 3.3. Policy specification mapping in fuzzy parameters

The administrative policy specification can be translated into management commands. As abstract rules are similar to the human feeling it is very difficult to map into computer rules, absolute and exact by nature. Fuzzy logic can taking care of informations with certain imprecise degree. Therefore, it is almost intuitive the approach of policy specification commands to fuzzy controller attributes. The event function can be represented by membership functions, while obligation function can be represented by rule base.

### 3.3.1. Mapping event command

The command **event** lists the events that fire obligation commands (**oblig**) or prohibition (**refrain**). An action can have many possible events, according to desired policy. The mapping of this command is a membership function with respective semantic values. Thus, the command **event** produces a membership function.

We show, in the listing 1, a specification using a command **event** and **oblig**. We attribute a value to events LD, MD and HD, meaning Low Delay, Medium Delay and High Delay. Those events will be used in the command **oblig** as a firing condition, in line 10, and the action *sched.increaseBW*, in line 11.

Listing 1: Mapping of event command

```
   event
 2     LD = 0.1 ;   // Low Delay
       MD = 0.5 ;   // Medium Delay
 4     HD = 0.9 ;   // High Delay

 6 inst
       oblig increaseSched {
 8         subject s ;
           target sched ;
10         on classEF.HD ;
           do sched.increaseBW()
12     }
```

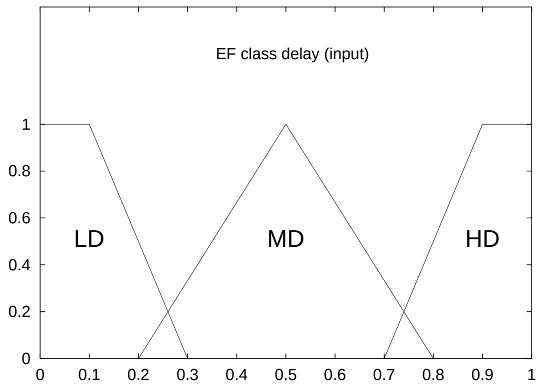


Figure 2: Membership function associate to event command

The membership function shape is shown in the figure 2. Note that each event will be associated to a semantic value of membership function. To be possible the mapping, we considered the event value is the central value of the semantic values and we refereed the geometric shape. The exact value and shape of those variables are not important, because they can be changed during GA optimization process.

### 3.3.2. Mapping oblig command

The command **oblig** indicates the obligation of an action execution if certain condition is satisfied (event). The mapping, therefore, is almost immediate. The parameter **on** of the command **oblig** is mapped in **if** parameter condition in rule base, and the parameter **do** is mapped in **then** parameter action.

We show, in listing 2, a specification using the command **oblig**. We considered the same membership function presented in the figures 2. In this command, case the event Medium Delay (MD) occurs in EF class (*classEF.MD*), the increasing the scheduler bandwidth action (*sched.increaseBW ()*) is fired.

Listing 2: Mapping of **oblig** command

```
inst
    oblig increaseSched {
        subject s ;
        target sched ;
        on classEF.MD ;
        do sched.increaseBW()
    }
```

The listing 3 shows a JFS[ 13] code of fuzzy controller rule base, based on policy specification shown in listing 2. We can see that the mapping of *increaseBW()* command was split in many *if <condition> then <action>* rules to cover all membership function values.

Listing 3: JFS code mapped from **oblig** command

```
program
    if classEF MD and sched LP then sched MP ;
    if classEF MD and sched MP then sched HP ;
    if classEF MD and sched HP then sched HP ;
```

## 4. PROTOTYPE SIMULATION

From the methodology proposed in the previous section, we can describe here the simulation environment and the implementation of a prototype to test the proposed methodology.

### 4.1. Simulation environment

The simulation tool used is the *ns* - Network Simulator, version 2.1b8[ 14]. The policy specification and verification tool was *Ponder Toolkit*[ 11], including a policy editor and language compiler. We used the version 1.0.1 of *Ponder Policy Editor* and version 0.2.1 of *Ponder Compiler*. The fuzzy library used in this work was developed with the JFS tool from Mortensen[ 13]. The C code generated by JFS toolkit is compiled and linked to *ns*.

### 4.2. Simulation topology

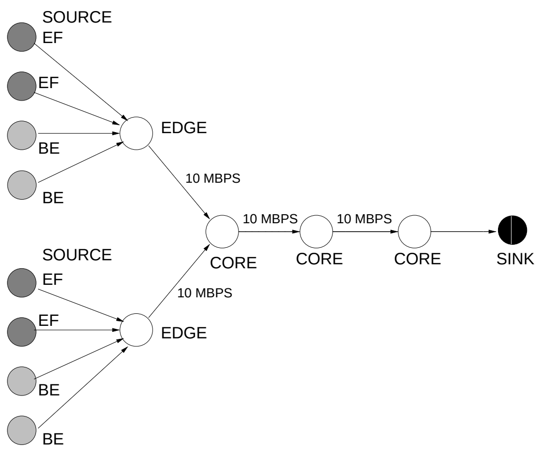
The evaluated topology is showed on figure 3a which shows a five nodes DiffServ domain, with two core nodes and three edge nodes. This topology gives two bottleneck for each flow, one in edge nodes and other in core nodes.

### 4.3. Simulation traffic

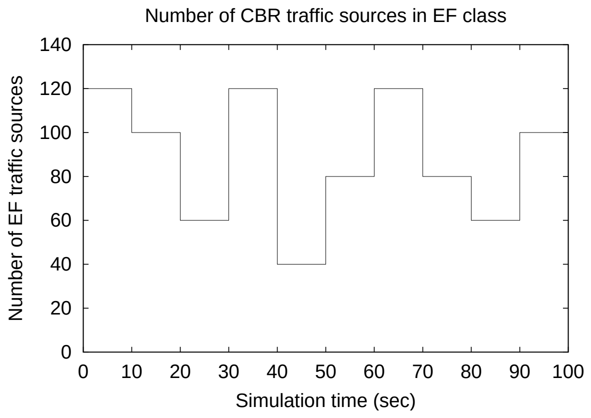
IP Telephony application was implemented with CBR and exponential On-Off traffics over UDP protocol. The CBR traffic has deterministic behavior and requires more bandwidth in network, while the On-Off traffic with exponential distribution is similar to a conversation. However, the On-Off average rate is smaller than CBR rate, then we added more On-Off traffic sources to obtain the desired traffic load in simulation. The voice traffic was classified in EF class[ 15] and competitive traffic in BE class, that is also CBR/UDP.

The figure 3b shows the number of simultaneous voice sources (EF class) during simulation (100 seconds). Each CBR voice source was defined with 64 Kbps, e.g., a PCM channel. Each On-off source, was 64 Kbps rate, with burst time of 400 ms and idle time of 600 ms, giving an average rate of 25,6 Kbps. The packet size used was 576 bytes.

The CBR traffic varied from 40 to 120 sources and the On-Off traffic varied from 100 to 300 sources. We used 300 competitive sources in BE class, with rate of 64 Kbps. The propagation time of each 10 Mbps link is 5 ms. All queues have 50 packets size, giving 23 ms of maximum delay per node.



(a) Simulation topology.



(b) Simulation traffic

Figure 3. Simulation environment

### 4.4. Conventional controller

To validate our proposal we defined, a conventional PD (Proportional Derivative) controller, because we always have better result with any dynamic provisioning mechanism when it is compared with a static provisioning scenario. The conventional controller calculates the adjusted line from the last three delays measures in class EF and apply this slope to the scheduler weight.

## 5. RESULTS

In this section we show the results of average delay and percentile, average jitter and percentile and drop rate of three situations: without controller, with conventional controller and with proposed fuzzy controller. All the simulations began allocating 50% of the bandwidth for each class. We started the measurements 5 seconds after starting the simulation to avoid empty network measurements.

The simulations used 1 second interval between samplings and controller actuation and the packet size was 576 bytes. We presented, in sections 5.2 and 5.1, a discussion about the controller behavior with operation interval and packet size variation.

The first measurement is the end-to-end delay in EF class. The table 1 shows the results of average, percentile 50, 90 and 95 of delay and jitter for CBR and On-Off traffics.

The table 2 shows the packets drop rate in EF and BE classes. We show the results of simulation with CBR and On-Off traffics without controller, using the conventional controller and using the fuzzy controller.

We can see an improvement in QoS (delay and jitter) and drop rate reduction in EF class with the fuzzy controller compared to without controller and conventional controller. Obviously, when the drop rate in EF class is reduced, the BE drop rate increase because network resources are exhausted. However, we can note that fuzzy controller produce

Table 1
Delay and jitter of the class EF in the simple topology (ms)

| Traffic | Average | | Percentile 50 | | Percentile 90 | | Percentile 95 | |
|---|---|---|---|---|---|---|---|---|
| | Delay | Jitter | Delay | Jitter | Delay | Jitter | Delay | Jitter |
| CBR NoCtrl | 43.4 | 1.7 | 43.0 | 0.4 | 65.8 | 0.4 | 75.3 | 7.3 |
| CBR ConvCtrl | 33.6 | 1.5 | 30.3 | 0.4 | 58.8 | 0.4 | 71.3 | 6.0 |
| CBR FuzzyCtrl | 19.3 | 1.0 | 17.3 | 0.4 | 34.9 | 0.4 | 41.1 | 0.4 |
| OO NoCtrl | 17.9 | 1.2 | 16.6 | 0.3 | 33.2 | 3.6 | 36.9 | 5.7 |
| OO ConvCtrl | 14.9 | 1.1 | 14.7 | 0.2 | 27.1 | 3.3 | 32.3 | 4.6 |
| OO FuzzyCtrl | 5.6 | 0.5 | 3.7 | 0.1 | 11.7 | 1.3 | 12.4 | 1.9 |

Table 2
Drop rate in EF and BE class

| Controller | EF (CBR) | BE (CBR) | Sum | EF(On-Off) | BE (On-Off) | Sum |
|---|---|---|---|---|---|---|
| No Ctrl | 0.0198 | 0.0766 | 0.0964 | 0.0379 | 0.1388 | 0.1767 |
| Conv. Ctrl | 0.0125 | 0.0843 | 0.0968 | 0.0272 | 0.1439 | 0.1711 |
| Fuzzy Ctrl | 0.0068 | 0.0795 | 0.0863 | 0.0027 | 0.1486 | 0.1513 |

lower aggregated rate, i.e., the sum of EF and BE drop rates is smaller than other summed rates, producing better general performance.

## 5.1. Packet size analysis

The first parameter analyzed was the packet size. We varied the packet size from 100 bytes to 576 bytes. We show percentile 90 delay for 100, 300 and 576 bytes packets. The controller interval was 1 second.

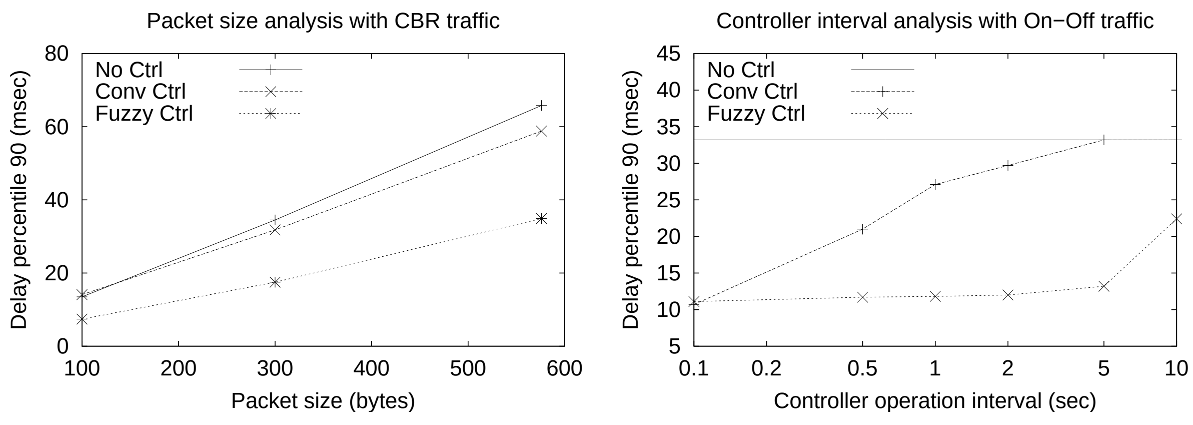
The CBR traffic evaluation, showed in figure 4(a), we can notice that the delays are similar with small packets (100 bytes), justified by the better scheduler with small packets, although fuzzy controller shows a small superiority.Finally, we concluded that the results obtained with fuzzy controller is better than the conventional controller and without controller for any packet size.

## 5.2. Controller operation interval analysis

The second parameter analyzed was the controller operation interval. The figure 4(b) shows the EF delay for On-Off traffic. We show the percentile 90 delay varying controller interval at 0,1 s, 0,5 s, 1 s, 2 s, 5 s and 10 s and the packet size was 576 bytes. To compare the results we show the percentile 90 delay without controller with a horizontal line.

Fuzzy controller response is better for any operation interval. This behavior is justified by the traffic variation profile every 10 seconds (see figure 3b). Only after 5 s we can notice a degradation in the fuzzy controller response.

Another conclusion is that fuzzy controller implements more efficient control than the conventional controller. Even recognizing more processing resources demanded by fuzzy controller, it can be used with larger intervals than the conventional controller to obtain equivalent results.



(a) Packet size CBR Traffic

(b) Controller interval On-Off Traffic

Figure 4. Packet size and controller operation interval analysis

## 6. CONCLUSIONS AND FUTURE WORKS

We presented in this work, an architecture of management system to improve QoS in a DiffServ domain. It was proposed, also, a methodology to mapping management policies in fuzzy controller parameters. This controller implements a dynamic provisioning mechanism, coordinated by policy based management system, improves the QoS in a DiffServ domain.

We made simulations changing the controller operation interval and the transmitted packets size. The analysis of controller interval showed that, with small interval, the fuzzy controller result is similar to conventional controller, becoming more significant with larger intervals. Comparing the results of CBR and On-Off traffic we noticed that the fuzzy controller is more efficient with On-Off traffic, that is the traffic found in real networks.

The analysis of packet size, showed that with small packets, the QoS metrics are similar for all the situations. Reduce the size of packets is an effective way to improve QoS, already known in ATM networks. Increasing the packet size, we observed a degradation in QoS in all situations, however the fuzzy controller continues showing best result. Again, we can see that fuzzy controller produces better comparative result to treat On-Off traffic comparing to CBR traffic, showing efficiency to deal actual traffics.

Finally, we can conclude that dynamic provisioning mechanisms shows advantages in maintain QoS when there are variations in traffic. The usual approach - over-provisioning - can offer QoS, however the cost of extra infrastructure is high. We also showed that the fuzzy logic seems appropriate to deal traffics with high variation, common in the actual situations. The proposed architecture is scalable and efficient.

As a future work, it should be done a performance evaluation, in the network equipments and in policy manager. As this work was made in simulation, it was not possible to evaluate the impact of the new mechanisms in a real world. We showed that fuzzy controller can obtain better QoS with larger intervals than conventional controller, but the computer overload would be studied.

## REFERENCES

1. S. Blake, D. Black, and M. Carlson, “An architecture for differentiated services,” RFC 2475, Dec. 1998.
2. Marcial Porto Fernandez, Aloysio de Castro P. Pedroza, and José Ferreira de Rezende, “Quality of service in a DiffServ domain using policy-based management,” in *XVII International Teletraffic Congress (ITC’17)*, Salvador, Brazil, Dec. 2001.
3. Marcial Porto Fernandez, Aloysio de Castro P. Pedroza, and José Ferreira de Rezende, “QoS provisioning across a DiffServ domain using policy-based management,” in *(Globecom 2001)*, San Antonio, USA, Nov. 2001.
4. C. C. Lee, “Fuzzy Logic in control systems: Fuzzy logic controller, Part II,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 2, pp. 419–435, Mar. 1990.
5. F. Herrera, M. Lozano, and J.L. Verdegay, “Tuning fuzzy logic controllers by genetic algorithms,” *International Journal of Approximate Reasoning*, vol. 12, no. 3, pp. 299–315, June 1995.
6. J. Velasco and L. Magdalena, “Genetic algorithms in fuzzy control systems,” in *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Periaux, M. Galan, and P. Cuesta, Eds., pp. 141–165. John Wiley & Sons, 1995.
7. Morris Sloman, “Policy driven management for distributed systems,” *Journal of Networking and Systems Management*, vol. 2, no. 4, pp. 333–360, 1994, Plenum Press.
8. DMTF, “Common Information Model (CIM) specification - version 2.2,” Distributed Management Task Force. http://www.dtmf.org/spec/cims.html, June 1999.
9. Marcial Porto Fernandez, Aloysio de Castro P. Pedroza, and José Ferreira de Rezende, “Optimizing fuzzy controllers with genetic algorithms for QoS improvement,” in *International Telecommunication Symposium (ITS’2002)*, Natal, Brazil, Sept. 2002.
10. Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman, “The Ponder policy specification language,” in *Policy 2001: Workshop on Policies for Distributed Systems ans Networks*, Bristol, UK, Jan. 2001, pp. 18–39.
11. Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman, “Ponder Policy Specification Language,” http://www-dse.doc.ic.ac.uk/policies/ponder.shtml.
12. E. Lupu and M. Sloman, “Towards a role based framework for distributed systems management,” *Journal of Network and Systems Management*, vol. 5, no. 1, pp. 5–30, Jan. 1997, Plenum Press Publishing.
13. Jan E. Mortensen, “JFS Fuzzy System,” http://www.inet.uni2.dk/ jemor/jfs.htm, 1998.
14. S. McCanne and S. Floyd, “ns Network Simulator - Version 2,” http://www.isi.edu/nsnam/ns/, 1998.
15. V. Jacobson, K. Nichols, and K. Poduri, “An expedited forwarding PHB,” RFC 2598, June 1999.